Implementation, Test und Dokumentation einer nutzerfreundlichen Schnittstellensoftware für den 3D-Scanner FASTRAK

Bachelorarbeit

Georg-August-Universität

Abteilung Ökoinformatik, Biometrie und Waldwachstum

Forschungsgruppe: Computer Graphics and Ecological Informatics

Verfasser:	Mgr. Ing. Katarzyna Wasilczuk
Matrikel-Nummer:	20851654
Betreuer:	Prof. Dr. Winfried Kurth
Zweitgutachter:	Dr. Jochen Schulz

Eingereicht am: 20. November 2012

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt zu haben. Die verwendete Literatur und sonstige Hilfsmittel sind vollständig angegeben.

Göttingen, 20. November 2012, ____

Danksagung

An dieser Stelle möchte ich mich bei Herrn Professor Dr. Winfried Kurth, Dr. Peter Surovy und Herrn Michael Henke für die Betreuung bedanken. Insbesondere danken möchte ich auch meinem Freund für die Unterstützung und den Rat und meinen Kommilitonen für die Hilfe bei den Vermessungen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

Es gibt im Allgemeinen keine frei verfügbare Software, die für das Vermessen von Pflanzen mit dem magnetischen 3D Scanner FASTRAK ausreichend gut geeignet ist. Die mit dem Scanner mitgelieferte Software dient vor allem dazu, schnell Konfigurationsdateien für spezielle Benutzerwünsche zu schreiben (Polhemus 2009). Für die Vermessung der topologischen Struktur von Pflanzen besser geeignet ist das Programm PiafDigit (Donès, Adam und Sinoquet 2006) (Abb. 1) der französischen Arbeitsgruppe UMR PIAF INRA-UBP (UMR PIAF INRA-UBP 2012).

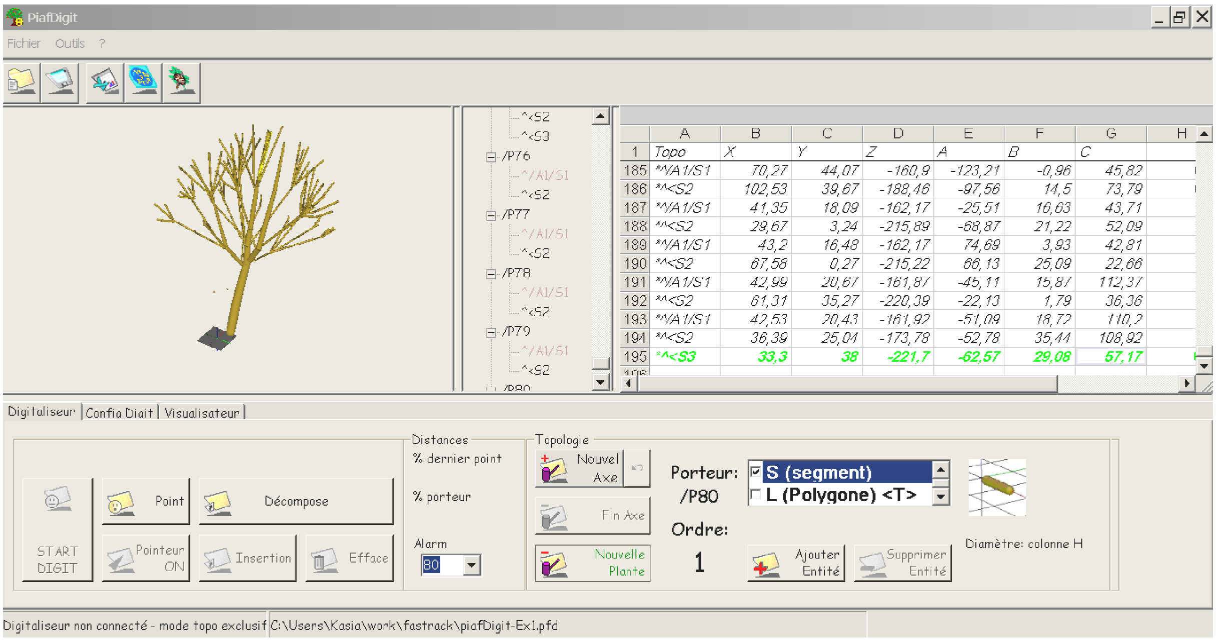


Abb. 1: Das Programm PiafDigit mit Datensatz von Dr. Peter Surovy.

Obwohl PiafDigit speziell zum Vermessen von Pflanzen entwickelt wurde, es ist in der Praxis nicht universell und zuverlässig einsetzbar. Zum einen ist die Oberfläche auf Französisch geschrieben, zum anderen treten regelmäßig Fehler im Speichermanagement und beim Behandeln von Ausnahmen auf. Deshalb besteht das Bedürfnis nach einem äquivalenten Ersatz, der eine ähnliche Funktionalität bietet. Der Entwurf und die Entwicklung einer Alternative zu Piafdigit ist Gegenstand dieser Arbeit. Das Ziel ist eine Implementierung, die dem Benutzer folgende Möglichkeiten anbietet:

- Aufnahme der topologischen Struktur einer Pflanze,
- Digitalisierung der Blätter, Blüten und Früchte,
- Vermessung des Volumens eines 3D Objekts,
- Eingabe des geographischen Richtung und der Neigung zu der XY-Ebene des Senders,
- Speicherung und Export der Informationen zu den gescannten Objekten,
- Kalibrierung des elektromagnetischen Feldes,
- Anwendung der Korrektur für die Messwerte nach einer Verschiebung des Senders,
- Verwendbarkeit sowohl unter Windows als auch unter Linux.

Die Funktionalitäten wurden im Rahmen der Arbeit ausführlich im Gelände und in geschlossenen Räumen anhand von verschiedenen Testobjekten geprüft. Die Testbeschreibungen sollen dem zukünftigen Benutzer auch als Handbuch zu der Software dienen. Die theoretischen und technischen Teile der Arbeit sollen den Einstieg für die nächsten Programmierer erleichtern. Das Kapitel *Hardware- und Softwarekomponenten* beschreibt die Grundlagen für die Entwicklung der Erweiterung (des Blender-Addons). Das Kapitel *Implementierung der Scan-Erweiterung* behandelt den theoretischen und technischen Hintergrund der Blender-Programmierung. Das Kapitel *Bedienungsanleitung für das Addon* beschreibt die zur Verfügung stehenden Funktionalitäten anhand von Beispielen. Das Kapitel *Einführung in Blender* hilft beim Einstieg in Blender. Die mit der neuen Software durchgeführten Testmessungen befinden sich im Anhang D (siehe Seite 139).

Kapitel 1

Hardware- und Softwarekomponenten

Das erste Kapitel ist als eine Zusammenfassung der Dokumentation der Hardware- und Softwarekomponenten gedacht. Deshalb sind die meisten Informationen, die der Leser hier finden kann, auch in den verlinkten Quellen zu finden. An manchen Stellen werden jedoch zusätzliche Informationen angegeben, die sich während der Entwicklungsarbeit angesammelt haben, aber nicht in der Dokumentation zu den Komponenten stehen. Der erste Teil des Kapitels beschreibt das elektromagnetische Trackingsystems FASTRAK, das für das Vermessen der Pflanzen verwendet wurde. Dabei werden nur die wichtigsten Informationen zur fehlerfreien Bedienung des Geräts ausgewählt. Die detaillierte Beschreibung aller Möglichkeiten der Hardware sind dem Benutzerhandbuch zu entnehmen (Polhemus 2009). Im zweiten Teil des Kapitels werden die Grundprinzipien der Programmierung mit Blender und dessen Erweiterung mit eigenen Skripten und Modulen beschrieben. Auch hier werden nur die für die Entwicklung relevanten Dinge detaillierter behandelt. Weiterführende Beschreibungen finden sich in den entsprechenden Quellen. Es wird bei den Erklärungen vorausgesetzt, dass der Leser ein Grundwissen über das Programmieren in Python besitzt. Für nähere Informationen siehe die Dokumentationsseite von Python (The Python Software Foundation 2011). Zum Abschluss des Kapitels werden die anderen Softwarebausteine, die in der Entwicklung der neuen Software benutzt wurden, erläutert. Es werden hier ein paar Tipps und Tricks, wie man die Installation am besten durchführt und überprüft, gegeben. Dabei werden die Unterschiede zwischen der Windows- und der Linux-Installation geschildert. Desweiteren wurden alle in dem Kapitel erwähnten Softwarekomponenten in einem Verzeichnis zu der neu entwickelten Erweiterung 3D_SCANN_ANWENDUNG\software im SVN-Archiv gesammelt, so dass man sie nicht unbedingt aus dem Internet selbstständig herunterladen muss. Allerdings sind zu jeder Komponente, soweit sie im Internet frei verfügbar ist, entsprechende Downloadlinks angegeben.

Trackingsystem Polhemus FASTRAK

Zur Vermessung einer Pflanze wird das elektromagnetische Trackingsystem FASTRAK benutzt. Das System ermöglicht dem Benutzer, die Position und die Orientierung eines Gegenstands im Messraum abzulesen. Die Information darüber ist mit sechs Freiheitsgraden (Abb. 2) für jeden an den Rechner geschickten Datensatz angegeben. Dabei beschreiben die drei ersten Zahlen den Ortsvektor, also die Translationen in der X-, Y- und Z-Achse des Empfängers in der Relation zum Sender, gefolgt von den drei Rotationen um die Z-, Y- und X-Achse, welche man als Gieren, Nicken und Rollen (eng. yaw, pitch, roll oder azimuth, elevation, roll) bezeichnet (Polhemus 2009). Wie bei Flugzeugen orientiert sich die positive Z-Achse auf dem Sender und dem Empfänger nach unten.

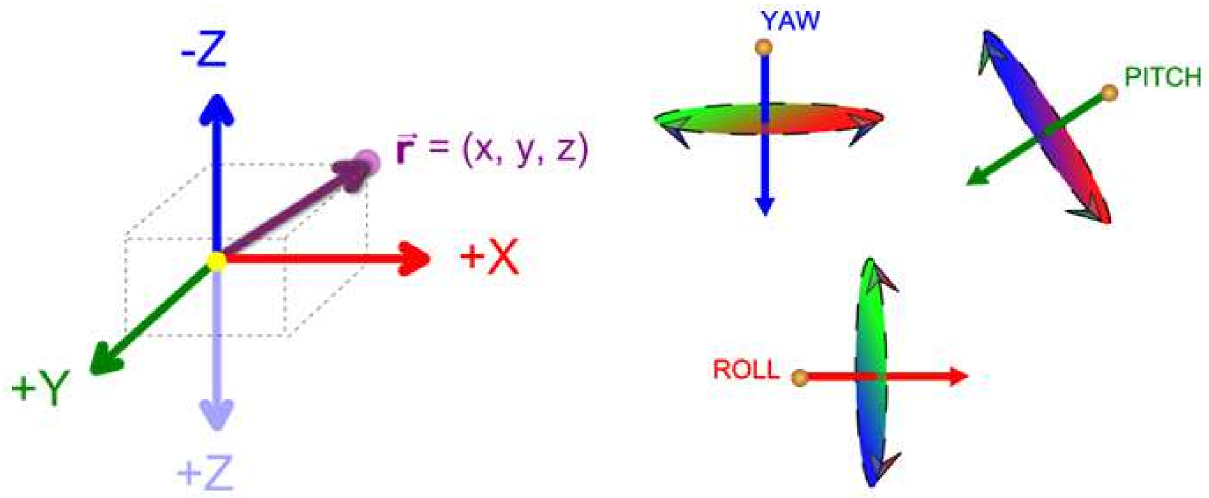


Abb. 2: Die sechs Freiheitsgrade der Scandaten. Links ist der Ortsvektor (Translation) in kartesischen Koordinaten abgebildet. Die Z-Achse ist so wie in dem FASTRAK-Trackingsystem dargestellt. Rechts befindet sich eine schematische Abbildung der Rotationen im 3D-Raum. Die X-Achse ist rot, Y-Achse grün und Z-Achse blau markiert. Die Halbebenen sind entsprechend rot-grün (XY-Halbebene), rot-blau (XZ-Halbebene) und grün-blau (YZ-Halbebene) markiert. Als YAW bezeichnet man die Rotation der XY-Ebene um die positive, senkrecht auf der XY-Ebene stehende Z-Achse (Gierachse). PITCH stellt die Drehung der XZ-Ebene um die senkrecht zur X-Achse stehende Y-Achse (Nickachse) dar. ROLL bezeichnet die Rotation der ZY-Ebene um die senkrecht stehende X-Achse (Rollachse).

Beim elektromagnetischen Tracking werden die Positionsinformationen mit Hilfe von drei senkrecht aufeinander stehenden Spulen generiert. Die Spulen befinden sich sowohl im Sender als auch im Empfänger. Nach Anlegen einer Wechselspannung im Sender fließt der Strom in drei Richtungen, die durch die Lage der Spulen festgelegt sind. Nachdem die Felder an den Empfänger übermittelt wurden, misst er für jeden der drei Ströme die entstehenden Magnetfelder bzw. die Stärke des induzierten Stromes. Damit kann das Gerät die Position und die Orientierung aus jeweils drei Messwerten pro Senderspule berechnen (Tegtmeier 2006) (Mauritczat 2003).

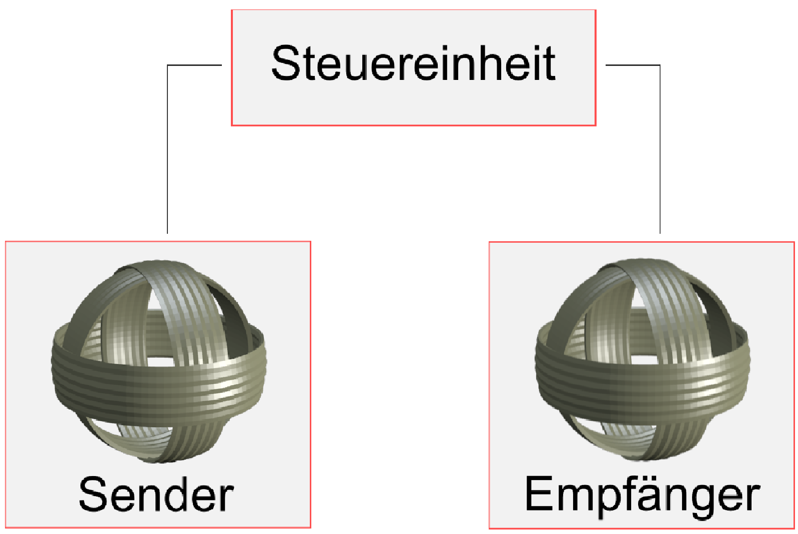


Abb. 3: Die senkrecht zueinander stehenden Spulen des Senders und Empfängers eines elektromagnetischen Trackingsystems und ihre Verbindung zur Steuereinheit.

Der Vorteil dieses Verfahrens im Vergleich zu photogrammetrischen Messungen ist, dass auch die verdeckten Teile der Pflanze gescannt werden können. Der Nachteil ist die Abnahme der Genauigkeit der Messergebnisse bei Störungen durch ferromagnetische Gegenstände und Geräte, die selbst ein elektromagnetisches Feld produzieren können, z. B. ein Handy oder ein Laptop.

Tab. 1: Ausgewählte Parametern des Polhemus FASTRAK-Scanners (SourVR 2008).

Parameter	Wert
Freiheitsgrade (DOF)	6
Maximale Anzahl der Empfänger	4
Reichweite	2 m
Maximale Messperiode	120 Hz (geteilt durch Anzahl der Empfänger)
Genauigkeit der Positionsauflösung	0.08 cm (Root mean square)
Genauigkeit der Rotationsauflösung	0.15 Rad (RMS)
Zeitliche Verzögerung	4 ms
Schnittstelle	RS-232 (Betriebssystem unabhängig) USB (nur Windows)
GUI/API Toolkit	Windows 2000/XP, 7

Komponenten des Scanners

Dieser Absatz ist eine Kurzfassung Zusammenfassung der für diese Arbeit relevanten Teile des Handbuchs zum Polhemus FASTRAK Scanner.

Die Hauptkomponenten des magnetischen Trackingsystems sind:

- die Steuereinheit mit dem Netzteil
- der Sender
- und der Empfänger.

Für das Verbinden mit einem Rechner stehen

- ein USB-Kabel,
- ein serielles Kabel
- und der Seriell-auf-USB-Adapter

zur Verfügung. Zu dem Scanner gehört auch eine Installations-CD mit einem USB-Treiber für Windows, einer Programmierschnittstelle und der zugehörigen

Dokumentation sowie einer Steuersoftware und einem Handbuch über die Bedienung des Trackingsystems (Anhang B, Seite 137). Die ausgewählten technischen Parameter, die den Trackingsystem beschreiben, sind in Tab. 1 zusammengefasst.

Die Steuereinheit

Die Steuereinheit kann überall an einem Arbeitsrechner platziert werden. Sie enthält die erforderlichen Ein- und Ausgangsanschlüsse und Bedienelemente.



Abb. 4: Steuereinheit des FASTRAK Scanners. Ansicht von vorne und hinten zusammen mit dem Netzteil für die Stromversorgung.

Es werden bis zu vier Empfänger unterstützt. Allerdings wurde für Entwicklungszwecke nur ein Empfänger benutzt, der immer an dem ersten Ausgang angeschlossen sein soll. Nur an diesem Eingang kann man die volle Funktionalität des Stifts ausnutzen. An der gleichen Seite der Steuereinheit befindet sich auch der Anschluss für einen einzelnen Sender und eine Powerleuchte. Der Anschluss für die serielle Schnittstelle RS-232 und USB Ausgänge, I/O Select und Anschlüsse für externe Synchronisation der Ein- und Ausgabe (Ext sync), Videosynchronisation (Video sync) und Powerschalter sowie der Stromeingang sind auf der anderen Seite der Steuereinheit angebracht. Die Steuereinheit ist nach dem Einschalten, wenn die Powerleuchte nicht mehr blinkt, betriebsbereit.

Der Sender

Der Sender muss vor dem Ausschalten der Steuereinheit angeschlossen werden, bevor sie angeschaltet wird, und abgetrennt werden, nachdem sie ausgeschaltet wurde. Die Steuereinheit soll auch nicht eingeschaltet werden, solange der Sender nicht angeschlossen ist. Beim Verlegen der Sender- und Empfängerkabel sollte man darauf achten, dass sie getrennt voneinander liegen. Bei Standardeinstellungen ist die Aufnahme der Position und der Orientierung des Empfängers auf der negativen Seite der X-Achse ausgeschlossen. Die Reichweite beträgt ca. 2 m in jede Richtung (vom Sender aus gesehen) (Polhemus 2009). Die Genauigkeit der Messung ist bis zu einem Abstand von ca. 70 cm gut, danach nimmt sie kontinuierlich wegen der magnetischen Verzerrung ab. Die Ursache der Störungen im elektromagnetischen Feld sind zusätzliche ferromagnetische Stoffe oder elektromagnetische Störfelder im Arbeitsvolumen (Siebert 2002) (Rosendahl 2003). Die Update-Rate für das Senden der Daten, wenn ein Empfänger angeschlossen ist, beträgt 120 Updates pro Sekunde (Polhemus 2009).



Abb. 5: Der Sender des elektromagnetischen Trackingsystems FASTRAK.

Der Empfänger

Der Empfänger ist wie ein Stift geformt, der auf einer Seite eine Taste hat (Abb. 6). Die Taste dient der Steuerung der Ausgabe des Scanners. Die Positionsmessungen erfolgen bei Standardeinstellung relativ zur Spitze des Stiftes. Der Empfänger kann an jedem der Empfängeranschlüsse an der Steuereinheit verwendet werden. Allerdings funktioniert die Taste nur dann, wenn der Stift an den ersten Eingang angeschlossen ist. Da die neue Software teilweise auch über den Stift gesteuert werden kann, muss bei allen Messungen darauf geachtet werden, dass der Empfänger mit der Steuereinheit mit dem ersten Anschluss verbunden ist. Desweiteren gilt für das Anschließen des Empfängers an die Steuereinheit das gleiche wie für den Sender. Das Kabel des Empfängers muss an die Steuereinheit angeschlossen werden, bevor sie angeschaltet wird, und abgetrennt werden, nachdem sie ausgeschaltet wurde. Falls man das Kabel abgetrennt hat, während die Steuereinheit eingeschaltet war, muss nach dem Anschließen des Empfängers das System neu initialisiert werden.



Abb. 6: Der Empfänger des elektromagnetischen Trackingsystems FASTRAK in Form eines Stiftes

Verbindung mit dem Rechner

Für das Anschließen an einen Rechner sind in der Steuereinheit zwei Ausgänge zur Auswahl vorgesehen. Für die Einstellungen der seriellen Schnittstellenparameter hat die Steuereinheit acht spezielle Schalter, die I/O Select (Abb. 7).

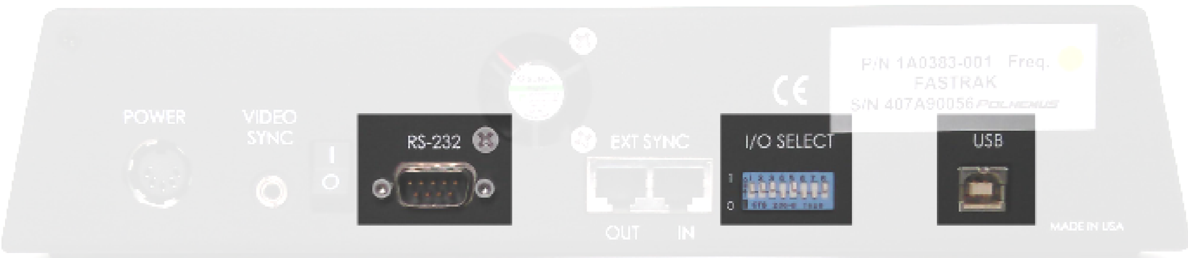


Abb. 7: Serieller Ausgang, I/O Select und der USB-Ausgang der Steuereinheit des Scanners der Firma Polhemus.

Für beide Anschlüsse stehen entsprechende Kabel zur Verfügung (Abb. 8). Da moderne Rechner nicht mehr mit einem seriellen Eingang ausgestattet werden, gibt es für das serielle Kabel einen Adapter, der die Verbindung mittels eines USB-Anschlusses des Rechners ermöglicht. Der Treiber für den Adapter ist im Lieferumfang enthalten und funktioniert sowohl unter Windows als auch unter Linux. Die USB-Schnittstelle des Scanners ist dagegen nur unter Windows verfügbar. Der Treiber für die Verbindung mittels USB wird von der Firma Polhemus nur für Windowssysteme geliefert. Deswegen wurde für die Kommunikation mit dem Gerät ausschließlich die serielle Schnittstelle RS-232 benutzt.



Abb. 8: Das serielle Kabel mit einem Adapter und das USB Kabel zur Verbindung des Scanners mit einem Rechner.

Die Steuereinheit besitzt auf der Rückseite eine Reihe von Schaltern (Abb. 7), die als I/O Select bezeichnet werden und der Steuerung der seriellen Schnittstelle RS-232 auf der Seite des Scanners dienen. Mit Hilfe der Schalter kann man an dem Gerät die Port-Parameter, die Baudrate, Zeichenbreite und Parität einstellen. Die Positionen der Schalter in der Standardeinstellung und ihre entsprechenden Funktionen sind in Tab. 2 aufgelistet. Die Einstellungen der Verbindung in der Software müssen den Einstellungen am Scanner angepasst werden.

Software zu dem Trackingsystem

Neben den Hardwareelementen liefert die Firma Polhemus eine Software, mit der man den Scanner ansteuern kann. Der Benutzer kann mit Hilfe des Programms FTGui unter anderem die Konfigurationsdateien für den Scanner erstellen und die gemessenen Koordinaten und Rotationen sehen (Abb. 9).

Es bietet aber für das Vermessen der Pflanzen nicht genügend Funktionalitäten. In der Entwicklung war es deshalb nötig, die Steuerbefehle aus selbst implementierter Software auf die serielle Schnittstelle zu schicken. Alle Befehle werden in Form von ASCII Zeichen an den seriellen Port gesendet. Das Handbuch beinhaltet eine detaillierte Beschreibung und Beispiele für jeden Befehlseinsatz. Die bei der Implementierung eingesetzten Befehle werden, zusammen mit einer kurzen Beschreibung, in Tab. 3 vorgestellt.

Tab. 2: Einstellungen, die von I/O Select geschaltet werden (Polhemus 2009), und deren Standardposition. Position unten entspricht der logischen 1 und oben der logischen 0.

Nr.	Gesteuerte Funktion	Standardposition	Wert
1	Baudrate	oben	
2	Baudrate	oben	115200 Hz
3	Baudrate	oben	
4	Nicht benutzt	unten	----------------------------
5	Zeichenbreite	oben	8 Bit
6	Parität	unten	None
7	Parität	unten	
8	Eingabe/Ausgabe	oben	RS-232

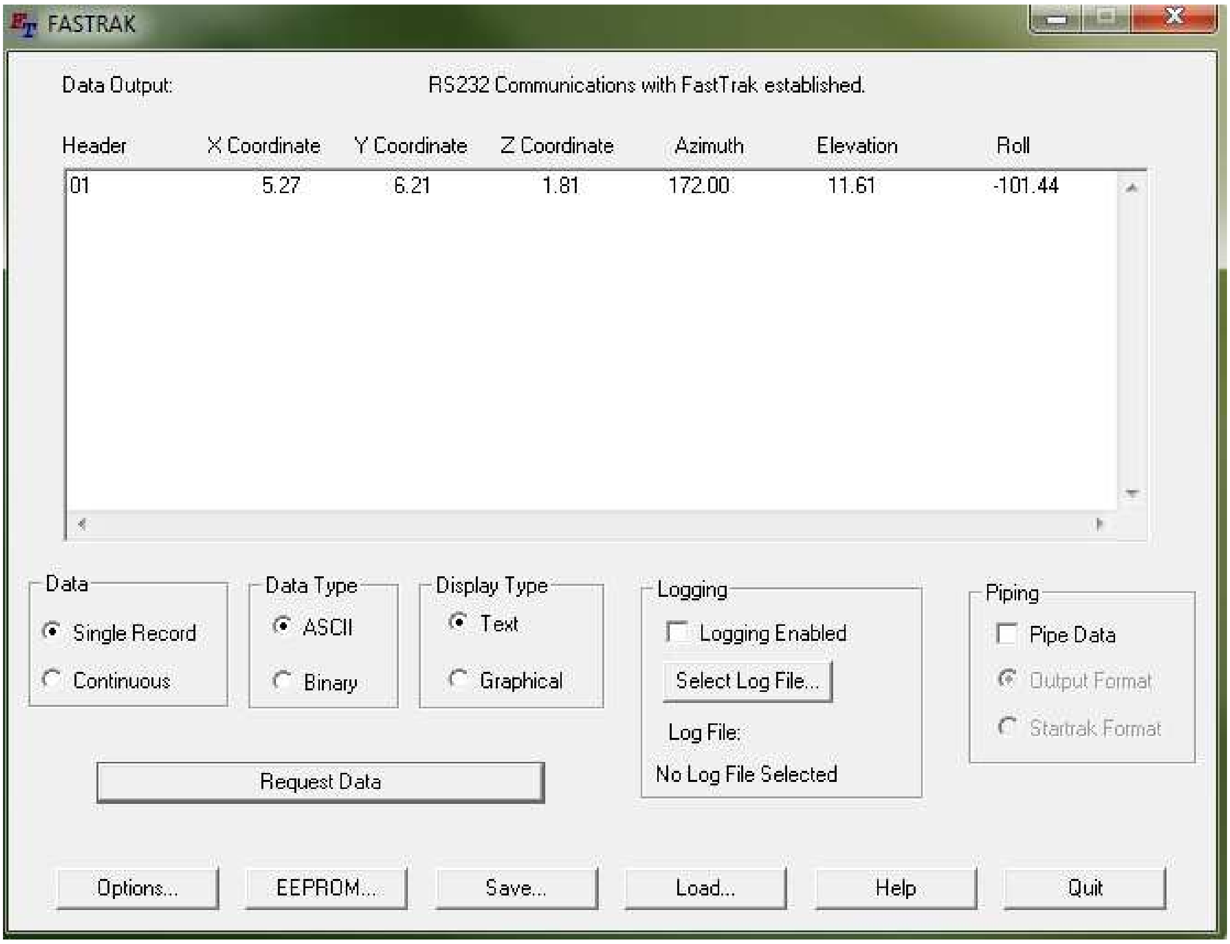


Abb. 9: FTGui – Benutzerinterface zum 3D Scanner der Firma Polhemus. In dem Textfeld ist eine Zeile der Ausgabe des Scanners dargestellt.

Tab. 3: Auswahl der Steuerbefehle für den FASTRAK Scanner (Polhemus 2009). Beachte Groß- und Kleinschreibung (die Zeichen <> stehen für einen Zeilenumbruch in dem Befehl).

Befehl	Beschreibung
C	Schaltet kontinuierliches Senden der Daten ein. Syntax: C<>
c	Schaltet das kontinuierliche Senden der Daten aus. Ein Datensatz wird nur dann gesendet, wenn er explizit angefordert wurde (Befehl P oder Drücken der Stifttaste). Syntax: c<>
e	Schaltet Stiftfunktion zwischen Maus und Punktmodus um. Syntax: e[Station],[Tastenmodus]<>
I	Inkrement für das Senden der Punkte. Das Inkrement muss bei dem Punktmodus zurück auf 0.0 gesetzt werden. Syntax: I[Station],[Abstand] <>
O	Ändert das Format der Ausgabe. Standardmäßig gehören zu einer Zeile der Ausgabe die xyz-Koordinaten und die drei Rotationswinkel in der Reihenfolge: yaw, pich, roll. Syntax: O[Station], [Befehl_1], ...,[Befehl_4] <>
P	Fordert eine Ausgabelinie an. Syntax: P<>
S	Fragt die Systemeinstellungen ab. Syntax: S<>
U	Wechselt die Einheit der Ausgaben in Inches. Syntax: U<>
u	Wechselt die Einheit der Ausgaben in Zentimeter. Syntax: u<>

Blender aus Sicht des Programmierers

Während des der Arbeit vorangegangenen Praktikums wurde festgestellt, dass die Möglichkeit der Verwendung schon fertiger und überprüfter Komponenten für die 3D Darstellung, die Elemente der Benutzeroberfläche und Programmverwaltung ohne Zweifel ein großer Vorteil für die Entwicklung ist. Aus diesem Grund wurde das neue Programm zur Vermessung der Pflanzen, genauso wie in dem Praktikum,

mit Hilfe der freien 3D-Modellierungssoftware Blender in Version 2.62 (Blender Foundation 2012) aufgebaut. Blender ist zwar nicht explizit dazu konzipiert, eine Interaktion mit dem Benutzer interaktiv abzuwickeln, liefert aber die Möglichkeit, sie zu ergänzen. Die Programmiersprache, in der die Ergänzungen geschrieben werden, ist Python 3.2 (The Python Software Foundation 2011). Python ist eine interpretierte Sprache. Blender hat einen eingebetteten Python-Interpreter, der mit Blender gestartet wird und aktiv bleibt. Dieser Interpreter bearbeitet die Skripte, die die Benutzeroberfläche zeichnen, und wird für einige der internen Blender-Werkzeuge verwendet. Die Anwendung wartet normalerweise während der Ausführung eines Skripts, bis die Verarbeitung zu Ende ist. In der Zeit wird nichts neu gezeichnet, und die Interaktion mit dem Benutzer ist auch angehalten. Die Verbindung mit der seriellen Schnittstelle und die gleichzeitige Interaktion mit dem Benutzer erfordert aber mehr Flexibilität. In Blender stellt das aber kein Problem dar, da die Skripte, die mit der Anwendung ausgeliefert werden, nur geringe Laufzeiten haben, so dass die 3D-Ansicht und die Interaktion mit dem Benutzer nicht blockiert werden. Wie man die Begrenzung umgehen kann, wird im weiteren Teil dieses Abschnittes beschrieben. Zuerst kommt eine kurze Einführung zu der Benutzung der Blender API (eng. application programming interface, deutsch „Schnittstelle zur Anwendungsprogrammierung“). Diese richtet sich nach den Informationen aus der Blender-Dokumentation (Blender Foundation 2012) und beabsichtigt nicht, jedes Thema abzudecken. Vielmehr ist der Zweck hier, den Leser mit den Teilen der API vertraut zu machen, die für die Implementierung gebraucht wurden. Zum Abschluss wird ein einfaches Beispiel einer Blender-Erweiterung, eines Addons, gegeben und erklärt, wie man externe Python-Module in dem Addon benutzen kann.

Blender Installation

Damit man mit der Blender API arbeiten kann, muss man Blender zuerst installieren und gestarten. Dies ist aber nicht weiter schwierig. Unter Windows gilt: Nachdem man den Installer heruntergeladen ist (Blender Foundation 2012), startet man ihn und folgt den Installationsschritten. Der Benutzer kann den Installer für Windows auch in dem SVN-Archiv zu der Anwendung finden (Anhang B, Seite 137). Falls die Anwendung mit dem Befehl blender aus der Konsole starten soll, ohne dass man den ganzen Pfad angibt, muss der Pfad zu dem Verzeichnis der blender.exe-Datei in die Umgebungsvariable Path in der Systemsteuerung von Windows angegeben werden (Abb. 13). Unter Linux ist es in dem Fall erforderlich, ein eigenes Verzeichnis für Blender zu haben. Man kann den Paketmanager nicht benutzen. Der Grund dafür ist, dass im Ubuntu Software Center die Version 2.52 zur Verfügung steht und nach dem Update dann die neuste Version von Blender. Der Addon wurde aber unter der Version 2.62 entwickelt. Es muss also ein entsprechendes Paket aus dem Internet geladen (Blender Foundation 2012) oder aus dem Archiv genommen werden (Anhang B, Seite 137). Anschleißend wird das Paket in das manuell angelegte Zielverzeichnis entpackt. So kann man mit der ausführbaren Datei aus dem Verzeichnis heraus die Anwendung starten. Wenn man Blender aus der Konsole, egal wo man sich gerade befindet, starten will, ist der beste Weg dies zu machen, einen Systemlink in dem /home/USR_NAME/bin

Verzeichnis zu der ausführbaren Datei der entsprechenden Version von Blender zu erstellen. Falls das Verzeichnis nicht existiert, legt man es einfach an. Der Grund, warum man das genau in dem Verzeichnis machen soll, ist, dass beim Starten Ubuntu die Datei ~/.profile einliest und in der Zeit die $PATH-Variable erstellt, die gebraucht wird, um die ausführbaren Dateien der Programme aus der Konsole zu starten. Die Einstellungen kann man mit gedit ~/.profile anschauen. Die Zeilen, die bei Start des Systems eingelesen und ausgeführt werden, sind unten dargestellt. Je nachdem, ob der Benutzer das Verzeichnis /home/USR_NAME/bin erstellt hat oder nicht, werden die Pfade aus dem Verzeichnis zu der Variable $PATH hinzugefügt oder nicht.

```
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

Beispielweise kann man festlegen, dass die ausführbare Datei für Blender mit dem Befehl blender262 starten soll. Dafür kann man z.B. Nautilus starten. Nachdem man entsprechende Verzeichnisse gefunden hat, zieht man mit der mittleren Maustaste die ausführbare Blender-Datei aus dem Blender-Installation-Verzeichnis in das /home/USR_NAME/bin-Verzeichnis und lässt einen Link mit dem gewünschten Namen erstellen (Abb. 10). Danach muss man das System neu starten und testen, ob der neuerstellte Link funktioniert. Falls alles richtig abgelaufen ist, startet Blender nach dem Eintippen des Befehls blender262, egal wie die Lokalisation gerade in der Konsole ist.

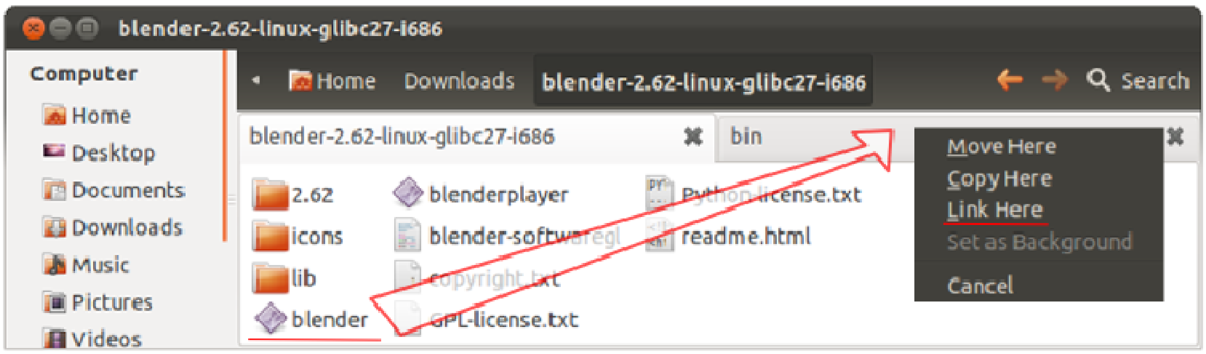


Abb. 10: Erstellung eines Links zu der ausfuhrbaren blender-Datei unter Linux. Die Datei wird vom Blender-Verzeichnis zu dem Zielverzeichnis mit der mittleren Maustaste herüber gezogen und verlinkt.

Blender API

Nachdem eine entsprechende Version auf beiden Betriebssystemen installiert wurde und man sie auch aus der Konsole ohne Schwierigkeiten starten kann, kann man sich dem eigentlichen Thema widmen – der Einarbeitung in die API von Blender. Wie man der Dokumentation (Blender Foundation 2012) entnimmt, ist die Blender API im Allgemeinen stabil. Mit jeder Blender-Version wird sie aber verbessert und unterscheidet sich deshalb von der letzten. Deshalb muss man unbedingt darauf achten, welche Versionen von Python und Blender passend zu

der API sind. Für den Scan-Addon wurde, gemäß der Python-Version von Blender, die Python-Version 3.2 (The Python Software Foundation 2011) und die Blender-Version 2.62 (Blender Foundation 2012) genommen. Der Dokumentation nach bietet die Schnittstelle folgende Möglichkeiten an:

- Zugriff auf interne Datenstrukturen der Anwendung (Szenen, Gitter, Partikel etc.)
- Änderungen der Benutzereinstellungen (Tastaturbelegung, Themen)
- Ausführung der Blender-Werkzeuge mit eigenen Einstellungen
- Hinzufügen neuer Elemente zu der Benutzeroberfläche (Menüs, Panels mit Buttons, Textfeldern, Auswahllisten etc.)
- Erstellung normaler und interaktiver Werkzeuge
- Erstellung von Rendering-Treibern, die mit Blender integriert werden können
- Definieren von neuen Einstellungen in vorhandenen Blender-Daten
- Zeichnen mit OpenGL-Befehlen von Python in der 3D-Ansicht

Vollständigkeit halber soll man auch die Begrenzungen der Blender/Python-API kennen. Mit der Bibliothek kann man nicht:

- neue Typen erstellen,
- benutzerdefinierte Eigenschaften jedem Typ zuweisen,
- definierte callback- und listener -Methoden benachrichtigen, wenn Daten geändert werden.

Beim Schreiben der Skripte für Blender muss man immer darauf achten, dem Python-Interpreter am Anfang des Skripts den Namen des Moduls, was den Zugriff auf die Blender-API ermöglicht, mitzuteilen. Dieses Modul heißt bpy und wird mit dem Befehl import am Anfang des Skriptes eingebunden (Abb. 11).

Die Skripte kann man in dem Texteditor der Anwendung über die Werkzeuge der Oberfläche oder über die Tastenkürzel ALT + P ausführen lassen. Alternativ kann man das Skript direkt in die eingebaute Python-Konsole einfügen. Die Prüfung der einzelnen Befehle der API in der Konsole ist ein guter Weg, um sich in die Schnittstelle einzuarbeiten. Darüber hinaus werden an jedem Button der Oberfläche Hinweise zu deren Python-Attributen und Namen angezeigt. Über den Rechtsklick auf einen Button oder einen Menüpunkt kann der entsprechende API-Dokumentationseintrag angezeigt werden. Blender hat auch ein eingebautes Logging-Fenster, das als ein Spickzettel beim Schreiben der Skripte dienen kann. In dem Fenster werden die Funktionenaufrufe angezeigt, die der Benutzer über Maus oder über einen Menüpunkt in der 3D-Ansicht ausführt und die als Operator in der API vorhanden sind.

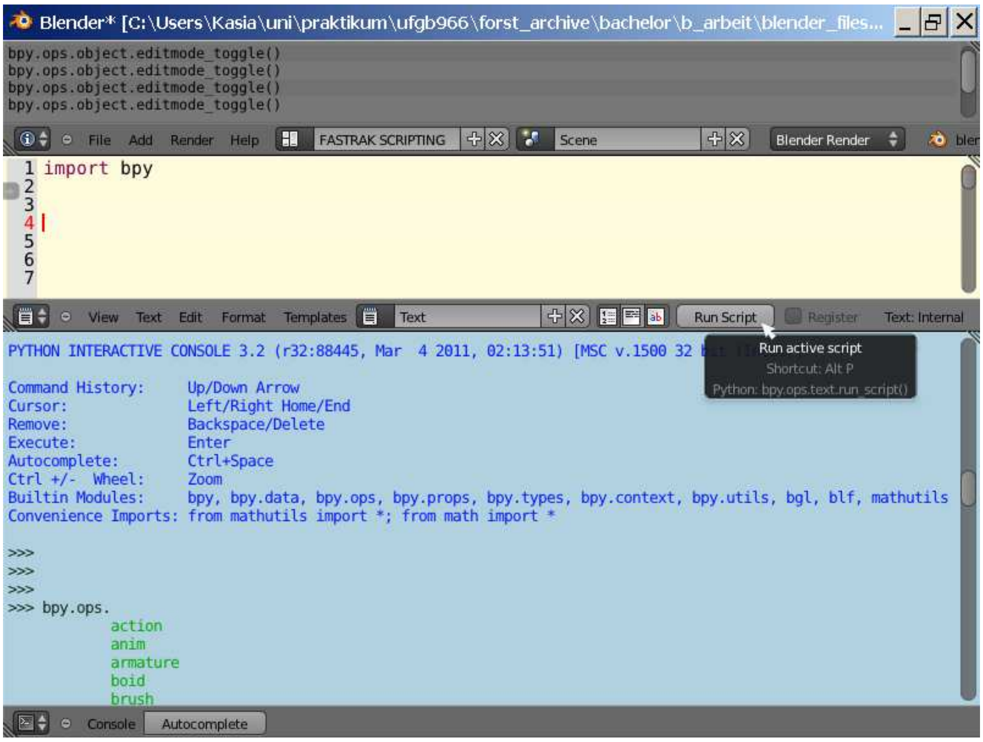


Abb. 11: Auswahl der Elemente aus der Benutzeroberfläche von Blender, die beim Schreiben der Skripte behilflich sind. Als erstes von oben befindet sich das Logging-Fenster (grau), in der Mitte ist der Texteditor (hell gelb) platziert, gefolgt von der Blender-Python-Konsole ganz unten (blau). Der Cursor zeigt auf den Button für das Starten der Skripte. In der Konsole nach Eintippen des Modulnamen bpy.ops gefolgt von einem Punkt, werden mit Hilfe der automatischen Vervollständigung weitere Eingabemöglichkeiten angezeigt.

Operatoren

Blender-Operatoren sind Werkzeuge (aus der Programmiersicht – ein Art von Klassen), auf die die Benutzer zugreifen, um mit der Anwendung interagieren zu können. Alle Operatoren werden in Python über das Modul bpy.ops aufgerufen. Man muss dabei beachten, dass die Operatoren gewisse Einschränkungen haben. Die wichtigsten davon sind:

- Man kann keine Datenstrukturen wie Objekte, Meshes, Materiale dem Operator übergeben. Stattdessen benutzen sie für die Manipulationen eine interne Struktur, den sogenannten Kontext (eng. context).
- Der Rückgabewert eines Operatoraufrufs ist nur eine Information über den erfolgreichen Abschluss. Es ist dabei egal, ob der Operator etwas gemacht hat oder keine Aktion durchgeführt wurde.
- Die poll-Funktion des Operators kann fehlschlagen, wenn eine API-Funktion eine Ausnahme mit den Details über die Ursachen auswirft.

Die poll-Funktion dient in den Operatoren zur Prüfung, ob die Maus in einem gültigen Bereich ist, oder ob das Objekt sich im richtigen Modus (Edit-Modus, Objekt-Modus, Textur-Modus usw.) befindet. Wenn die Bedingungen nicht erfüllt sind, liefert die Funktion False zurück, sonst ist die Rückgabe der Funktion True.

```
>>> bpy.ops.object.editmode_toggle.poll()
False

>>> bpy.ops.object.editmode_toggle.poll()
True
```

Das Resultat des Aufrufes der poll-Methode wird noch anhand eines Beispiels näher betrachtet. Es wird der Operator bpy.ops.object.editmode_toggle() für Ein- oder Ausschalten des Edit-Modus genommen. Wenn alles richtig abgelaufen ist und der Operator normal beendet wurde, sieht man in der Blender-Konsole den Rückgabewert {'FINISHED'} oder einen ähnlichen String.

```
>>> bpy.ops.object.editmode_toggle()
{'FINISHED'}
```

Wenn kein Objekt zum Editieren in der 3D-Ansicht vorhanden ist, verursacht der Aufruf eines Operators in der Python-Konsole eine Fehlermeldung. Die Skripte, die solche Fälle nicht abfangen, werden auf der Stelle abgebrochen.

```
>>> bpy.ops.object.editmode_toggle()
Traceback (most recent call last):
  File "<blender_console>", line 1, in <module>
  File "C:\Program Files (x86)\Blender Foundation\Blender\2.62\scripts\modules\bpy\ops.py"
, line 180, in __call__
    ret = op_call(self.idname_py(), None, kw)
RuntimeError: Operator bpy.ops.object.editmode_toggle.poll() failed, context is incorrect
```

Derartige Fehlermeldungen können sehr oft während der Interaktion des Skripts mit einem Benutzer vorkommen. Um sie zu vermeiden, wurde deshalb in dem Quellkode des Addons, wo es nötig war, mit den try/except-Klauseln gearbeitet. Zusätzlich wurde auch die Möglichkeit des Aufrufes der poll-Funktionen der Operatoren ausgenutzt, um zu überprüfen, ob er im aktuellen Kontext ausgeführt werden kann oder nicht. In dem Beispiel wurde die if/else-Klausel eingesetzt.

```
>>> if bpy.ops.object.editmode_toggle.poll():
...     bpy.ops.object.editmode_toggle()
... else:
...     # handle this exception
```

Die ersten Informationen, wie man die Operatoren aus der Blender-API in Python benutzt, sind jetzt bekannt. Es wurde aber noch nichts darüber gesagt, wie man eigene Operatoren in Blender definiert und benutzt. Im weiteren Verlauf des Abschnittes wird erklärt, wie man einen einfachen Operator schreibt, der die Namen aller einer Szene sichtbaren Objekte auf der Systemkonsole ausgibt. Der Quellkode zu dem Beispiel befindet sich in Anhang 1 (Seite 133).

In Python wird dafür einfach eine Klasse definiert, die von der Blender-Klasse bpy.types.Operator erbt. Der Kommentar unter der Definition des Klassennamens wird als ein Hinweis (eng. Tooltip) zu dem Operator in der Benutzeroberfläche angezeigt.

```
class PrintNamesOperator(bpy.types.Operator):
    '''Tooltip: Prints visible objects names to the console.'''
```

Als nächstes werden zwei Blender-Variablen, bl_idname und bl_label, erstellt.

```
    bl_idname = "scene.simple_names_print_operator"
    bl_label = "Simple Print Operator"
```

Die erste Variable dient dazu, einen Namen des Operators in der Blender-Umgebung zu definieren. Später kann man den Operator mit bpy.ops.scene.simple_print_names_operator von Blender aus aufrufen. Bei der Definition von bl_idname ist zu beachten, dass der String einen Punkt enthalten muss und keine Leerzeichen oder Sonderzeichen enthalten darf. Die zweite Variable wird in Blender als die Überschrift eines Buttons in der Benutzeroberfläche zu sehen sein. Blender hat mehrere Sätze von derartigen bl_-Variablen für jede Klasse aus der API. Sie sind in der API-Referenz bpy.types.Operator dokumentiert (Blender Foundation 2012). Desweiteren wird festgelegt, dass der neue Operator nur dann ausgeführt werden darf, wenn die Anzahl der sichtbaren Objekte größer Null ist.

```
    @classmethod
    def poll(cls, context):
        return len(context.visible_objects) > 0
```

Wenn das der Fall ist, wird der Rückgabewert der poll-Funktion True sein, und als nächstes wird die execute-Methode des Operators ausgeführt. Im anderen Fall wird eine Fehlermeldung, ähnlich der aus dem Beispiel zu der poll-Funktion, generiert. Als nächstes wird in der execute-Methode eine Schleife über alle sichtbaren Objekte der Szene (context.visible_objects) erstellt. In jeder Iteration wird der Name des Objekts (obj.name) im Terminal ausgeben.

```
    def execute(self, context):
        for obj in context.visible_objects:
            print(obj.name)
        return {'FINISHED'}
```

Der Operator ist jetzt fertig. Er ist aber noch nicht in Blender integriert. Das Laden der eigenen Blender-Klassen geschieht mit den Blender-eigenen Methoden register und unregister.

```
def register():
    bpy.utils.register_class(PrintNamesOperator)


def unregister():
    bpy.utils.unregister_class(PrintNamesOperator)
```

Diese Funktionen sollen normalerweise am Ende des Skripts platziert werden. Der Aufruf der register- und unregister-Methoden ermöglicht auch ein Ein- und Ausschalten der Addons und das Starten von neuen Skripten, während Blender läuft. Beim Ausführen des Skripts im Texteditor wird die register-Funktion nicht automatisch ausgeführt. Deshalb wird am Ende die main-Funktion der Anwendung aufgerufen.

```
if __name__ == "__main__":
    register()
```

Nach dem Starten des Skriptes kann der Operator jetzt sowohl aus der Blender-Python-Konsole als auch in der 3D-Ansicht über das Drücken der Leertaste und das danach eingeblendete Kontextmenü aufgerufen werden. Zum Testen des Operators wurde im Texteditor in einer neuen Datei eine if/else-Klausel eingeführt, die eine Fehlermeldung, falls die poll-Methode False zurückgegeben hat, verhindert.

```
# test call
if bpy.ops.scene.simple_names_print_operator.poll():
    bpy.ops.scene.simple_names_print_operator()
else:
    print('There are no visible objects in the scene!')
```

Somit, falls der Benutzer ein paar Objekte in der 3D-Ansicht platziert, die nicht versteckt sind, wird eine Liste der Namen der Objekte im Terminal ausgegeben. Zum Testen werden zuerst drei Objekte in Blender erstellt – eine Lampe, eine Kamera und ein Würfel. In der Terminalausgabe werden nach dem Aufruf des einfachen Operators die Namen der drei Objekte angezeigt.

```
C:\Users\Kasia>blender
found bundled python: C:\Program Files (x86)\Blender Foundation\Blender\2.62\python
Cube
Lamp
Camera
```

Wenn bei dem Test gerade kein Terminal geöffnet ist, kann man eines über den Menüpunkt Help > Toggle System Console anzeigen lassen. Für den zweiten Test wurden alle Objekte in der 3D-Ansicht versteckt und dann der Operator noch ein Mal aufgerufen. Wie erwartet, sieht man in dem Terminal die Information, dass es keine sichtbaren Objekte in der Szene gibt, und der Operator verursacht keine Fehlermeldung.

```
C:\Users\Kasia>blender
found bundled python: C:\Program Files (x86)\Blender Foundation\Blender\2.62\python
There are no visible objects in the scene!
```

UI-Gestaltung

Für die Erstellung der 3D-Scan-Anwendung fehlt noch die Kommunikation mit dem Benutzer über die UI-Elemente (eng. user interface). Dazu benötigt man zwei weitere Teile der Blender API. Zum einen, wie über Python die Komponenten der Benutzeroberfläche erstellt werden, und zum anderen, wie mit Hilfe der API die Attribute, bzw. Eigenschaften, die zusätzliche Informationen zu den Blender-Datentypen enthalten, definiert werden. Es wird dazu das Operatorskript aus dem ersten Beispiel erweitert. Es wird ein Blender-Panel mit einem Button für Benutzung des Beispieloperators erzeugt. Desweiteren wird in dem Panel eine Eigenschaft der Szene, die in einer Klasse definiert wird, platziert. Sie soll dem Benutzer dazu dienen, den Operator-Button aus- und einzublenden. Es soll ebenfalls möglich sein, aufzuzählen, wie viele Male die Eigenschaft ihren Zustand geändert hat, und die Information darüber in dem Panel anzuzeigen (Abb. 12).

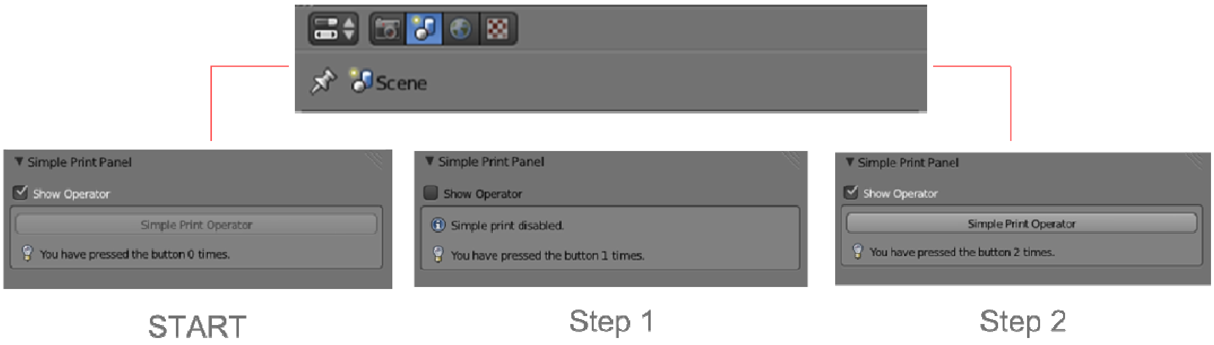


Abb. 12: Das Panel zu dem einfachen Operator befindet sich in dem Reiter Scene **des** Properties-**Fensters. Abgebildet sind die Startansicht und die zwei nächsten Zustände nach dem Aus- und wieder Einblenden des Operator-Buttons.**

Der Quelltext der Operatorklasse bleibt hier so, wie er in dem ersten Beispiel definiert war. Es werden zwei weitere Blender-Klassen hinzugefügt. Eine für das Zeichnen des Panels, die von der Klasse bpy.types.Panel erbt, und eine, die den Schalter für das Anzeigen und Verstecken des Operator-Buttons erstellt. Die zweite Klasse erbt von der Blender-Klasse bpy.types.PropertyGroup.

```
class PrintNamesOperator(bpy.types.Operator):
    # body of the class ...

class SimplePrintProperty(bpy.types.PropertyGroup):
    # body of the class ...
```

In der letzten Klasse wird nur ein einziges Attribut, eine boolesche Variable show, die in einer Blender-Struktur bpy.props.BoolProperty abgekapselt ist, erstellt.

```
class SimplePrintProperty(bpy.types.PropertyGroup):
    show = bpy.props.BoolProperty(name = "Show Operator",
                                  description = "Show/hide the operator button.",
                                  default = True,
                                  update = update_count_func)
```

Blender hat Mechanismen, um solche Konstrukts in dem UI als passende Benutzerflächenelemente ohne größeren Aufwand zeichnen zu lassen. Die Blender-BoolProperty wird z. B. in dem Interface als eine Checkbox erscheinen. Das Attribut name dient zu Beschriftung der Checkbox. Das Attribut description wird von Blender im Inhalt des Hinweises zu dem UI-Element angezeigt. Der Standardwert der Eigenschaft wird über das Attribut default definiert. Das Attribut update dient der Übergabe der Funktion, die bei jeder Änderung des Zustands der Eigenschaft aufgerufen werden soll. Man schreibt hier nur den Namen der entsprechenden Funktion. Auf die Art und Weise sind in Blender einfachere Python-Typen wie String, Integer oder Float abgekapselt. Kompliziertere Elemente der Benutzeroberfläche, wie z.B. Comboboxen oder Gruppen von Radiobuttons, kann man mit der EnumProperty erstellen. Gruppen von Bool-, Integer- oder Float-Variablen werden als BoolVektorProperty, IntVektorProperty oder FloatVektorProperty deklariert. Im Tutorium aus der Dokumentation zu Blender (Blender Foundation 2012) gibt es eine genauere Beschreibung mit Beispielen.

Als nächstes soll das boolesche Attribut der Klasse SimplePrintProperty auch eine update-Methode besitzen. Der Name der Funktion update_count_func wurde schon bei der Erstellung der booleschen Variable show zum Attribut update der Eigenschaft hingeschrieben. Die Funktion wird am Anfang des Skripts platziert.

```
def update_count_func(self, context):
    context.scene.my_count = context.scene.my_count + 1
    return None
```

Die Definition dieser Funktion muss nicht unbedingt am Anfang stehen. Sie muss aber auf jeden Fall vor der Definition der Eigenschaft stehen, die die Methode benutzen soll. Sonst gibt es Fehlermeldungen während der Registrierung, da der Interpreter die entsprechende update-Methode nicht findet. Desweiteren dürfen die update-Funktionen immer nur zwei Variablen für die Übergabe haben. Die erste Variable erhält eine Referenz auf die Instanz der Klasse, von der sie aufgerufen wird, und muss self genannt werden. Die zweite Variable bekommt die Referenz auf den Blender-Kontext und muss context heißen. Der Rückgabewert muss dabei immer None sein. Die Funktion wird die Änderungen des Zustandes der Checkbox aufzählen. Die Information über die Anzahl der Änderungen wird in einer IntProperty my_count, die im Rumpf der Klasse SimplePrintPanel deklariert ist, abgespeichert.

```
    bpy.types.Scene.my_count = bpy.props.IntProperty(default = 0)
```

Jetzt gibt es alles, was man zum Aufbauen des Beispielpanels braucht. Man kann also die Klasse, die für das Zeichnen der Elemente der Benutzeroberfläche des Panels gebraucht wird, fertigschreiben. Die Klasse erbt von der Blender-Klasse bpy.types.Panel.

```
class SimplePrintPanel(bpy.types.Panel):
    bl_label = "Simple Print Panel"
    bl_idname = "OBJECT_PT_simple_panel"
    bl_space_type = "PROPERTIES"
    bl_region_type = "WINDOW"
    bl_context = "scene"
```

Im Vergleich zu der Definition eines einfachen Operators aus dem ersten Beispiel ist es hier erforderlich, ein paar weitere Blender-Klassenvariablen hinzuzufügen. Die zusätzlichen bl-Variablen definieren die Lage des Panels in der Benutzeroberfläche. Das Panel wird im Eigenschaftenfenster (bl_space_type, bl_region_type) in dem Tab scene (bl_context) platziert.

Die wichtigste und meistens einzige Funktion, die in der Panelklasse gebraucht wird, ist draw. Die ausführliche Beschreibung aller Befehle und Parameter, die bei der Programmierung der Benutzeroberfläche helfen, findet der Leser in der Dokumentation zu Blender (Blender Foundation 2012). Es wird hier eine Zusammenfassung gegeben, so dass der Rumpf der neu geschriebenen draw-Methode für den Leser verständlich ist. Zum Einordnen der UI-Elemente wird ein layout-Attribut der Blender-Klasse bpy.types.Panel benutzt.

```
    def draw(self, context):
        col = self.layout.column()
        col.prop(context.scene.my_simpe_print, "show")
        box = col.box()
        if context.scene.my_simpe_print.show:
            box.operator("scene.simple_names_print_operator")
        else:
            box.label(" Simple print disabled.", icon = "INFO")
            box.label(" You have pressed the button %s times." %
context.scene.my count,
```

Die einfachste Methode, die Komponenten in die Benutzeroberfläche einzuordnen, sind die Befehle:

- layout.prop(…),
- layout.operator(…),
- layout.label(…).

Desweiteren können die Elemente zeilenweise mit den Befehlen:

- row = layout.row(…),
- row.prop(…),
- row. operator (…),
- row.label(…)

oder auch spaltenweise mit:

- col = layout.column(),
- col.prop(..),
- col. operator (…),
- col.label(…)

usw. platziert werden. Für kompliziertere Einordnungen gibt es in der API den Befehl layout.split. In der draw-Funktion steht außer den aufgezählten Elementen des Layouts ein nächstes Element, eine Box (col.box). Die Box entspricht einer Spalte. Dank ihres helleren Hintergrundes gruppiert und setzt sie optisch die UI-Komponenten (Abb. 12) ab. Damit nicht nur die Oberfläche gut aufgeräumt ist, sondern auch der Quellkode dazu, gibt es in der Blender-Dokumentation ein paar Hinweise, wie man die draw-Methode klar und effizient strukturiert. Als erstes soll man bei der Programmierung immer wieder versuchen, die gleichen Variablennamen in der ganzen Funktion zu benutzen, z. B.

- row für Zeilenlayout
- col für Spaltenlayout
- split für die aufgeteilte Einordnung
- sub z. B. für eine Spalte in der Spalte usw.

Man soll auch vermeiden, dass sich in den Panel-Klassen rechenaufwendige Operationen befinden. Die UI-Klassen sind einfach aufgebaut und sollen so bleiben. Der Grund dafür ist, dass in Blender die Panels ständig neu gezeichnet werden. Längere Berechnungen in einem Skript können den Prozess deutlich verlangsamen und damit auch die Arbeit mit der Anwendung unkomfortabel

machen. Weiterhin ist es sinnvoll für die Verwaltung der Benutzeroberfläche und deren Elemente, die poll-Methode in den Blender-Klassen zu definieren. Es ist oft nötig, dem Benutzer zu signalisieren, dass ein Button oder eine andere Komponente der Oberfläche gerade nicht aktiv ist. Üblicherweise geschieht das durch das Ausgrauen des entsprechenden Elements. Blender übernimmt die Verwaltung darüber, sobald in der Operatorklasse oder Eigenschaftklasse die poll-Funktion definiert ist. Es gibt auch eine Möglichkeit, das Ausgrauen mit Hilfe der layout-Parameter selber zu steuern. In dem Falle bleiben aber die Eigenschaften und Operatoren aktiv, obwohl sie ausgegraut sind, und der Programmierer muss dafür sorgen, dass die Abläufe in seinem Skript richtig bleiben. Wenn die poll-Funktion in einer Panelklasse definiert ist, stellt sie sicher, dass das Panel nur in einem bestimmten Kontext gezeichnet wird und sichtbar ist und sonst nicht.

Es sind jetzt alle Eigenschaften, Operatoren, Klassen und Methoden, die zur Erfüllung der Anforderungen des Beispiels nötig sind, definiert. Der letzte Schritt ist die Integration der neuen Klassen in Blender. Dies wird mittels der register- und unregister-Methoden gemacht, genauso wie das in dem ersten Operatorbeispiel vorgeführt wurde.

```
def register():
    bpy.utils.register_class(SimplePrintProperty)
    bpy.utils.register_class(PrintNamesOperator)
    bpy.utils.register_class(SimplePrintPanel)

    bpy.types.Scene.my_simpe_print = bpy.props.PointerProperty(type =
SimplePrintProperty)

def unregister():
    bpy.utils.unregister_class(SimplePrintProperty)
    bpy.utils.unregister_class(PrintNamesOperator)
    bpy.utils.unregister_class(SimplePrintPanel)
```

Der einzige Unterschied hier ist ein zusätzlicher Befehl für die Initialisierung der Eigenschaftenklasse SimplePrintProperty. Nach dem Aufruf der register-Funktion in der main-Funktion von Blender kann das neue Panel in dem Reiter Scene des Eigenschaftenfensters angeschaut und getestet (Abb. 12) werden. Der ganze Quellkode des zweiten Beispiels findet sich in Anhang 1 (Seite 133).

```
if __name__ == "__main__":
    register()
```

Addon-Erstellung

Es ist jetzt bekannt, wie man Skripte für Blender schreibt und wie man sie integriert. Die Vorgehensweise ist aber nur für Testzwecke oder für den eigenen Gebrauch zu empfehlen. Wenn das Skript für dritte Personen verfügbar sein soll, ist die bessere Wahl, eine richtige Erweiterung für Blender, ein sogenanntes Addon, zu schreiben. Im Verlauf dieses Abschnittes wird dazu ein Beispiel erstellt, das auf den zwei vorherigen Beispielen aufbaut. Der Quelltext, der zum Ende des zweiten Beispiels angefertigt wurde, wird fast komplett übernommen. Dabei werden die register- und unregister-Funktionen geändert und ein Addon-Header

am Anfang des Skriptes eingefügt. Danach wird noch das Addon-Skript in das richtige Verzeichnis der Blender-Installation kopiert.

```
if __name__ == "__main__":
    register()
```

Der Addon-Header besteht meistens aus zwei Teilen. Der erste Teil ist der GPL-Lizenz-Text. Blender ist konzipiert als eine frei verfügbare Anwendung, und die meisten Skripte und Addons werden auch als solche verbreitet. Der Text der Lizenz befindet sich in jedem Addon, die mit Blender ausgeliefert wird. Wenn die Software kommerziell genutzt werden soll, muss man sich mit der Blender Foundation (Blender Foundation 2012) in Verbindung setzen, um die Bedingungen zu klären.

```
#====================== BEGIN GPL LICENSE BLOCK ======================
#
#  This program is free software; you can redistribute it and/or
#  modify it under the terms of the GNU General Public License
#  as published by the Free Software Foundation; either version 2
#  of the License, or (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to the Free Software Foundation,
#  Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
#
#======================= END GPL LICENSE BLOCK ========================
```

Zum zweiten Teil des Addons-Headers gehört die Definition der Blender-Variable bl_info. Es ist ein einfaches Python-Dictionary, das ein paar festgelegte Einträge beinhalten soll. Diese verwendet Blender zur Beschreibung des Addons in der Benutzeroberfläche des File > User Preferences …-Fensters in dem Reiter Addons (Abb. 39).

```
bl_info = {"name": "Simple Print Names Addon",
           "author": "Katarzyna Wasilczuk",
           "version": (1, 0),
           "blender": (2, 6, 2),
           "location": "Properties Window > Scene",
           "description": "Text abouth this addon.",
           "warning": "e.g. dependieces ...",
           "wiki_url": "",
           "tracker_url": "",
           "category": "3D View"}
```

Die meisten Einträge der Liste sind eindeutig, deshalb wird nicht explizit auf sie eingegangen. Unter dem Namen version soll eine Versionsnummer des Addons stehen. Danach steht die Versionsnummer von Blender. Der letzte Eintrag in dem Dictionary ist category. Hier kann der Programmierer entscheiden, zu welcher Kategorie das Addon gehört. Die Kategorien des Addons-Reiters helfen dem Benutzer, die Anzahl der angezeigten Addons schnell zu begrenzen. Wenn der

Eintrag leer bleibt oder die Kategorie nicht existiert, werden die Informationen zu dem Addon nur in der Kategorie *All* eingeblendet.

Die letzte Anpassung, die vorgenommen wird, ist eine Vereinfachung der *register-* und *unregister*-Funktionen des Skripts aus dem zweiten Beispiel (Anhang 1, Seite 133). Statt jede Klasse des Addons einzeln in den Methoden aufzuzählen, was bei größeren Projekten sehr aufwendig wäre, werden zwei andere Operatoren, die Mehrfachregistration ermöglichen, verwendet. Sie haben im Namen den Suffix *_module* statt *_class*.

```
def register():
    bpy.utils.register_module(__name__)

    bpy.types.Scene.my_simpe_print = bpy.props.PointerProperty(type =
SimplePrintProperty)


def unregister():
    bpy.utils.unregister_module(__name__)
```

Damit ist das Addon fertig. Bevor man aber es im Fenster *File > User Preferences … >Addons* sehen kann, muss man das Skript einmal in das Blender-Addon-Installationsverzeichnis kopieren. Unter Windows sieht der Pfad meistens folgendermaßen aus:

```
'C:\path_to_installation_directory\Blender
```

Das Installationsverzeichnis heißt hier meistens *Program Files*. Unter Linux, wenn Blender über den Paketmanager installiert wurde, befindet sich das Addon-Verzeichnis meistens an folgender Stelle:

```
'/usr/share/blender/2.63/scripts/addons'
```

Nachdem das Addon-Skript an die richtige Stelle kopiert wurde, erscheint in Blender im Fenster *File > User Preferences … >Addons* der Addon-Eintrag mit allen Informationen, die in dem *bl_info*-Dictionary enthalten waren, zusammen mit einer Checkbox, die das Addon lädt und abmeldet (Abb. 39). Das Addon ist immer dann verfügbar, wenn die Checkbox angekreuzt ist. Nach dem Registrieren des Addons über die Checkbox in dem Fenster *File > User Preferences … >Addons* wird das gleiche Panel mit den gleichen Funktionalitäten, wie es schon im zweiten Beispiel beschrieben wurde, erscheinen. Der ganze Quellkode der Erweiterung ist im Anhang 1 (Seite 134) zu finden.

Externe Python-Module für das 3D-Scan-Addon

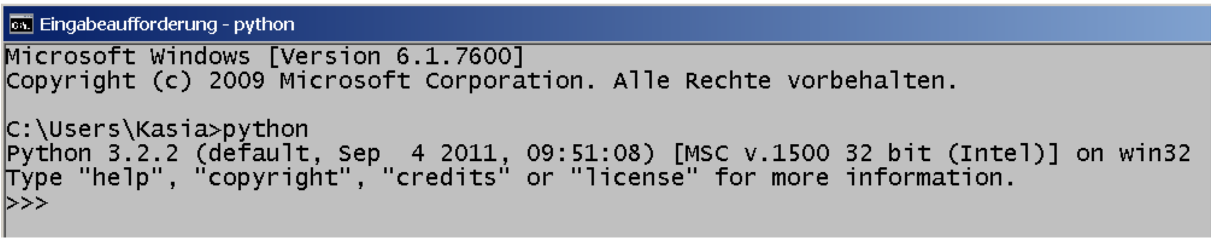
Es wurden schon fast alle Komponenten, die man in der Entwicklung braucht, besprochen. Da Blender schon einen Python-Interpreter mitliefert, wurde auf die Installation einer externen Python-Version noch nicht eingegangen. Es werden aber für das Scannen noch zusätzliche Python-Module, die nicht im Umfang der standardmäßigen Blender-Installation enthalten sind, gebraucht. Das erste Modul ist *pyserial*, das die Funktionalitäten zur Abwicklung der seriellen Kommunikation

anbietet. Die zwei nächsten Module, numpy und scipy, werden in den Berechnungen der Delaunay-Triangulation beim Aufbauen der Gitter verwendet. Das letzte externe Modul wird für das Abspielen der Tonsignale bei einer Messung gebraucht. Die Töne sollen einem Benutzer, der keinen Bildschirm sieht, helfen, den aktuellen Zustand der Anwendung zu kontrollieren und ihm zu folgen. Unter Windows wird dafür das Modul winsound benutzt. Für die Linux-Version des Addons wurde das Python-Modul pygame gewählt.

Python 3.2

Damit man die aufgezählten Module in Blender integrieren kann, braucht man zuerst eine externe Python-Installation, mit deren Hilfe die Module auf dem Arbeitsrechner vorbereitet werden. Sehr wichtig ist dabei, dass die Versionsnummer der externen Python-Installation mit der Version von Python in Blender übereinstimmt. In der Entwicklung wurde Blender 2.62 mit Python 3.2 benutzt. Demzufolge wurde sowohl unter Windows als auch unter Linux Python 3.2 installiert.

Für Windows gibt es einen Python-Windows-Installer. Die entsprechende Version kann aus dem Internet geladen werden (The Python Software Foundation 2011) oder alternativ, da die gesamte externe Software im Verzeichnis der neuen Anwendung gesammelt ist (Anhang B, Seite 137), kann man den Installer aus dem SVN-Archiv benutzen. Der Ablauf einer standardmäßigen Installation ist sehr einfach. Die einzelnen Schritte, die der Benutzer während der Installation machen muss, sind in Anhang C (Seite 138) beschrieben. Nachdem der Installer seine Arbeit abgeschlossen hat, muss noch die Umgebungsvariable in den Systemeigenschaften aktualisiert werden (Abb. 13). Sobald der Pfad zum Installationsverzeichnis von Python in der Variablen abgespeichert wurde, kann man das Terminal mit dem Befehl python den Python-Interpreter starten.



Wenn keine Fehlermeldung zu sehen ist, bedeutet das, dass alles richtig abgelaufen ist. Als nächstes kann man sich jetzt mit der Installation der zusätzlichen Python-Module beschäftigen.

Der Ablauf der Installation der richtigen Python-Version unter Linux hängt von der Linux-Distribution ab. Bei einigen ist sie über den Paketmanager erreichbar, bei den anderen nicht. In der Entwicklung wurde Linux Ubuntu 11.10 ‘‘Oneiric Ocelot‘‘ für 32 Bit Architekturen benutzt. Die Installations-CD für das Betriebssystem ist im Anwendungsverzeichnis im SVN-Archiv abgespeichert (Anhang B, Seite 137) oder kann aus dem Internet heruntergeladen werden (Ubuntu 2011). Bei der Version von Linux kann man Python 3.2 entweder mit Hilfe des Ubuntu Software Center

(Abb. 14) oder in der Konsole mit dem Befehl *sudo apt-get install python3-all-dev* installieren.

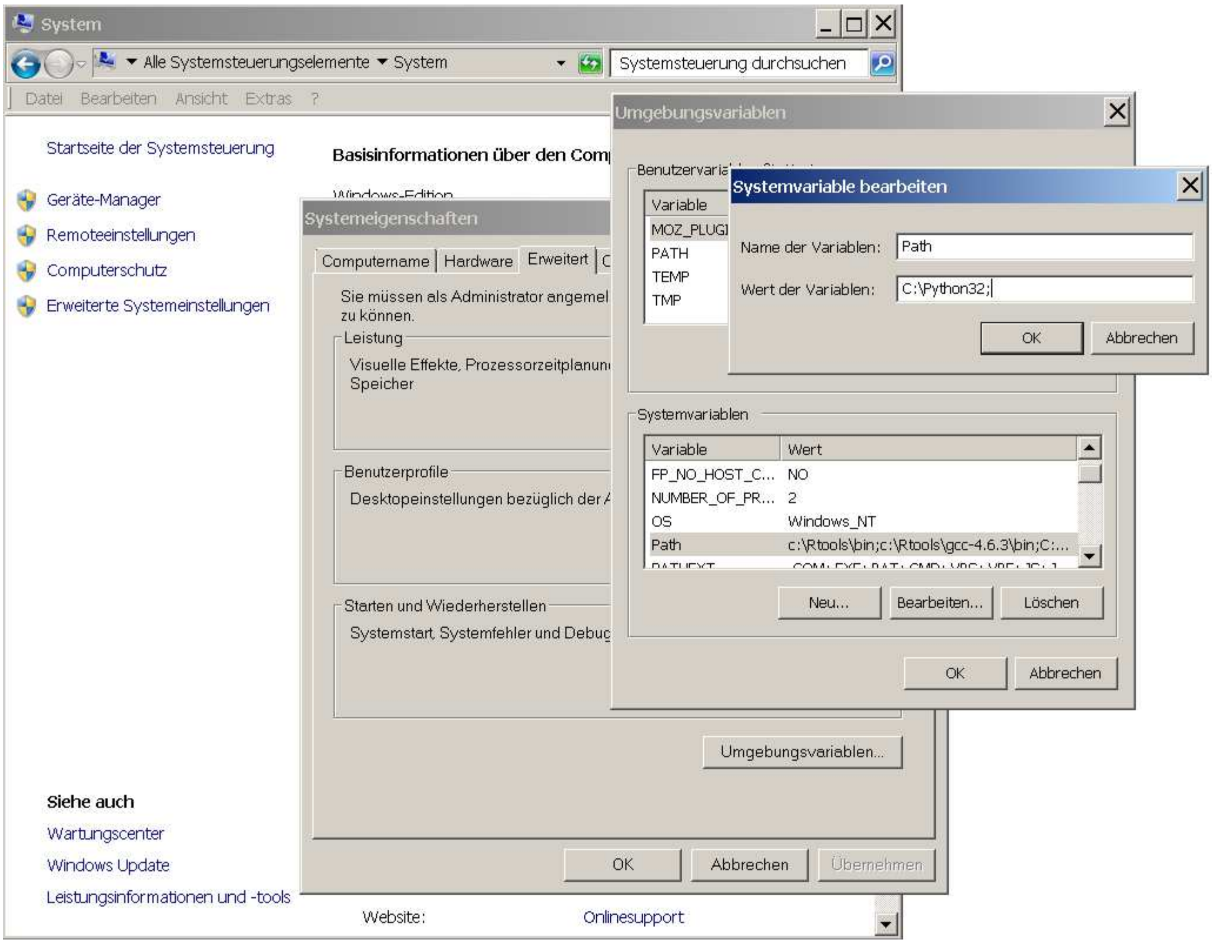


Abb. 13: Aktualisierung der Windows-Umgebungsvariable path**. Der Benutzer schreibt den Pfad zum Installationsverzeichnis der aktuellen Version von Python.**

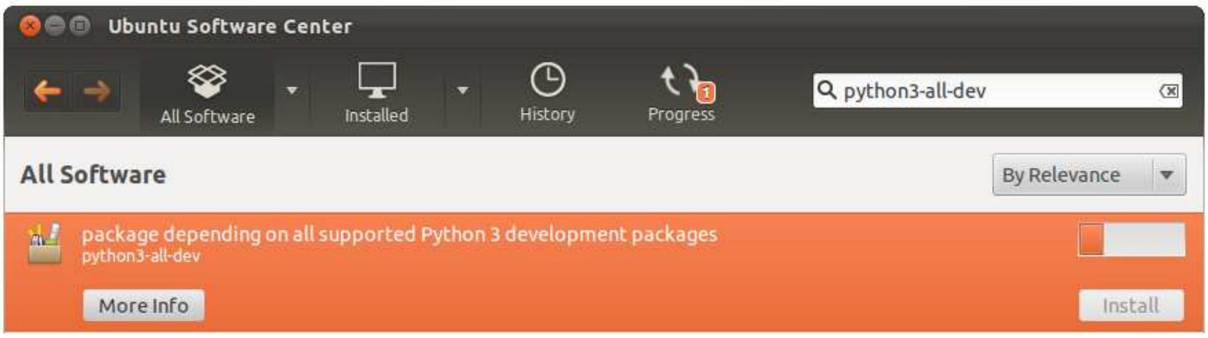


Abb. 14: Installation von Python 3.2 unter Linux Ubuntu Software Center**.**

Das hat den Vorteil, dass eventuelle Abhängigkeiten automatisch erkannt und mit installiert werden. Alternativ gibt es eine Möglichkeit, die ganze Installation unabhängig von den von Linux angebotenen Python-Quellen durchzuführen. Den komprimierten Python-Quellcode findet man auf der Website (The Python Software Foundation 2011) oder im vorbereiteten Verzeichnis im SVN-Archiv (Anhang B,

Seite 137). Allerdings müssen dann bei der Installation alle Konfigurationen selber festlegt werden. Danach kann man den Quellcode bauen und installieren. Andere dabei benötigte Software muss dann manuell nachinstalliert werden. Die Details dazu befinden sich in der Readme.txt-Datei des benutzten Installationspakets. Wenn die Installation abgeschlossen ist, kann man in der Systemkonsole, ähnlich wie unter Windows, mit dem Befehl python3 testen, ob sie erfolgreich abgelaufen ist. Die Installation ist richtig, wenn der Befehl für das Starten von Python keine Fehlermeldung verursacht und die Versionsnummer gleich 3.2.2 ist.

```
kasia@speedy:~$ python3
Python 3.2.2 (default, Sep  5 2011, 22:09:30)
[GCC 4.6.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Das Modul für die serielle Kommunikation – pyserial

Der 3D-Scanner kommuniziert mit dem Rechner über eine serielle Schnittstelle. Die Schnittstelle zum Benutzer, Blender, besitzt aber keine eingebauten Python-Module für die serielle Kommunikation. Dieser Mangel wurde mit der Installation des Moduls pyserial behoben. Sowohl unter Windows als auch unter Linux ist es nicht weiter kompliziert. Die entsprechenden Quellen befinden sich im SVN (Anhang B, Seite 137) oder können aus dem Internet geladen werden (Liechti 2010). Unter Windows muss der Benutzer den Installer für pyserial starten und sich durch die Installation führen lassen (Anhang C, Seite 138). Nach einer erfolgreichen Installation kann man ohne Fehlermeldung mit dem Befehl import serial das Modul in der Python-Konsole laden.

```
C:\Users\Kasia>python
Python 3.2.2 (default, Sep  4 2011, 09:51:08) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import serial
>>>
```

Unter Linux muss der Benutzer nach dem Herunterladen (Liechti 2010) (SVN - Anhang B, Seite 137) und Entpacken der Quellen für pyserial zuerst in das Installationsverzeichnis in der Konsole wechseln.

```
cd /path/to/pyserial-2.5
```

Danach wird das Modul mit dem Befehl build gebaut. Da am Anfang bei der Python-Installation das Entwicklerpaket gewählt wurde, sollten dabei keine Schwierigkeiten auftreten.

```
python3 setup.py build
```

Wenn das Bauen erfolgreich durchgeführt wurde, kann man den Prozess mit dem Befehl install abschließen.

```
sudo python3 setup.py install
```

Zum Testen, ob die Installation wirklich gelungen ist, wird das Modul mit dem Befehl import, genauso wie unter Windows, importiert. Fehlerfreies Laden des Moduls bedeutet einen Erfolg. Man sollte dabei nicht vergessen, das Installationsverzeichnis zuerst zu verlassen. Andernfalls wird der Befehl für das Laden fehlschlagen.

```
kasia@speedy:~$ python3
Python 3.2.2 (default, Sep  5 2011, 22:09:30)
[GCC 4.6.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import serial
```

Als nächstes werden die Grundlagen der Programmierung einer seriellen Schnittstelle anhand eines Beispielskripts geschildert. Eine vollständige Beschreibung der Möglichkeiten des Moduls findet der Leser auf der Dokumentationsseite im Internet (Liechti 2010). In dem Beispiel wird an die serielle Schnittstelle eine Nachricht geschickt, mit der der Status des Scanners abgefragt wird. Der Quellcode zu dem Beispiel befindet sich in Anhang A (Seite 136). Bevor man mit dem Programmieren startet, muss man herausfinden, wie die Namen der Ports unter Windows und unter Linux lauten. Alle anderen Einstellungen werden den Werten, die auf dem Scanner festgelegt wurden (Tab. 2), angepasst. Mit dem Befehl MODE.COM prüft man unter Windows, ob serielle Ports zur Verfügung stehen. Meistens ist das der Port COM1. Wie es aber aus der Ausgabe des Befehls hervorgeht, heißt in diesem Falle der serielle Port COM16.

```
C:\Users\Kasia>MODE.COM

Status von Gerät COM16:
-----------------------
    Baudrate:         1200
    Parität:          None
    Datenbits:        7
    Stoppbits:        1
    Timeout:        OFF
    XON/XOFF:         OFF
    CTS-Handshaking:  OFF
    DSR-Handshaking:  OFF
    DSR-Prüfung:      OFF
    DTR-Signal:       ON
    RTS-Signal:       ON


Status von Gerät CON:
---------------------
    Zeilen:           300
    Spalten:          130
    Wiederholrate:    31
    Verzögerungszeit:1
    Codepage:         850
```

Bei der Suche nach dem Port unter Linux kann beispielsweise das Befehl dmesg | grep tty benutzt werden.

```
kasia@speedy:~$ dmesg | grep tty
[    0.000000] console [tty0] enabled
[    1.073484] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
```

Wie man der Ausgabe des Befehls entnehmen kann, heißt der serielle Port unter Linux anders als bei Windows. Somit weiß man, dass der volle Name des seriellen Ports unter Linux in diesem Fall /dev/ttyS0 ist. Die Unterschiede werden in dem

Beispiel berücksichtigt. Es wird jeweils, bevor der Name des Ports für die serielle Verbindung gesetzt wird, die Plattform abgefragt. Dafür werden, zusätzlich zu dem serial-Modul, die zwei Funktionen platform und search aus zwei Python-Modulen, sys und re, am Anfang des Skripts importiert. Sie sind standardmäßig in der Python-Installation enthalten und brauchen nicht extra installiert werden.

```
>>> import serial
>>> from sys import platform
>>> from re import search
```

Nachdem alle Funktionen, die gebraucht werden, geladen sind, wird mit der if/else-Klausel eine Instanz der Klasse Serial erstellt. Dabei muss man unbedingt beachten, dass im Falle einer seriellen Verbindung über einen Port nur eine Instanz kommunizieren darf.

```
>>> if platform == 'win32':
...     ser = serial.Serial('COM16')
...
>>> if search('linux', platform):
...     ser = serial.Serial('/dev/ttyS0')
```

Falls der Kanal schon belegt ist, wird beim Zugriff darauf eine Fehlermeldung generiert und die Verbindung wird nicht erstellt. Als nächstes wird getestet, ob die Verbindung auf dem Port geöffnet ist. Falls sie noch geschlossen ist, wird sie mit dem Befehl ser.open geöffnet. Sonst kann nichts auf die serielle Schnittstelle geschickt oder von ihr empfangen werden.

```
>>> if not ser.isOpen():
...     ser.open()
... else:
...     print('\nThe connection is opened on port = %s\n'%ser.port)
...

The connection is opened on port = COM16
```

Zur Sicherheit wird noch die neue Instanz der Klasse Serial auf der Konsole ausgegeben. Der Wert des Attributes port stimmt mit dem Wert des Ports aus der Ausgabe des Befehls MODE.COM überein, dh. das Testbeispiel läuft gerade unter Windows. Gemäß Tab. 2 sind die zwei anderen Variablen bytesize und parity passend zu der Hardware-Einstellung gesetzt.

```
>>> ser
Serial<id=0x24cfe90, open=True>(port='COM16', baudrate=9600, bytesize=8, parity='N',
stopbits=1, timeout=None, xonxoff=False, rtscts=False, dsrdtr=False)
```

Deshalb muss man nur noch das Attribut baudrate auf 115200 Baud setzen und die Konfiguration der Verbindung ist mit den Einstellungen auf dem Scanner abgeglichen.

```
>>> ser.baudrate = 115200
```

Damit ist aber noch nicht alles fertig. Bevor man den Status des Scanners abfragen kann, muss noch das Attribut des ser-Objektes, das timeout, gesetzt werden. Es ist für das Beenden des Wartens auf eine Antwort aus der seriellen Schnittstelle verantwortlich. Der Standardwert des Attributes ist None, d.h. in dem Falle gibt es

keine Zeitbegrenzung für die Antwort des Gerätes. Falls eine Frage an die serielle Schnittstelle geschickt wurde, wartet die Instanz, bis sie die Antwort empfangen hat. So kann ein Skript, das z.B. in Blender eingebunden wurde, die ganze Anwendung blockieren. Solche Situationen müssen vermieden werden. Wenn das timeout zu klein gewählt ist, kann es passieren, dass statt der Ausgabe nur eine leere Linie empfangen und ausgegeben wird, obwohl das Gerät eigentlich die Daten weggeschickt hat. Nach ein paar Proben wurde festgestellt, dass ein ausreichend langer Wert für das timeout in diesem Beispiel 0.1 Sekunden ist.

```
>>> ser.timeout = 0.1
```

Zur Überprüfung wird noch einmal die ser-Instanz ausgegeben. Alle Werte wurden gemäß der Tab. 2 gewählt. Die Attribute stopbits, xonxoff, rtscts und dsrtr können bei den Standardeinstellungen gelassen werden.

```
>>> ser
Serial<id=0x24cfe90, open=True>(port='COM16', baudrate=115200, bytesize=8, parity='N'
, stopbits=1, timeout=0.1, xonxoff=False, rtscts=False, dsrdtr=False)
```

Jetzt ist alles vorbereitet, um den Scanner mit dem Befehl ser.write und ser.readline nach seinem Status zu fragen. Wenn das Gerät an den Rechner angeschlossen und eingeschaltet ist, wird eine Statusinformation ausgegeben, die ähnlich wie die Ausgabe unten aussieht. Auf diese Weise kann man leicht die Kommunikation verifizieren.

```
>>> ser.write(bytes('S', 'latin1'))
1
>>> ser.readline()
b'21S3F0  0      2.0.3 Polhemus Fastrak - USB Support \x00\r\n'
```

Die 2 steht für den Typ des aufgenommenen Wertes. An der nächsten Stelle befindet sich die Nummer der Station (1). Danach folgen drei Zeichen (3F0) eines hexadezimalen Codes der Konfiguration. Als nächstes steht ein Fehlerstatus. Falls es einen Systemfehler gibt, befindet sich an dieser Stelle eine andere Zahl als Null. Zum Abschluss sind die Informationen über die Version des Produktes und ein Umbruchzeichen platziert.

Die Module für numerische Berechnungen - numpy und scipy

Für die neue Software wird eine Methode, bzw. eine Software, zur Rekonstruktion einer Oberfläche aus einer 3D-Punktwolke gebraucht. Blender ist sehr vielseitig, besitzt aber keinen eingebauten Generator für Oberflächengitter. Die Zeitbegrenzung erlaubte es nicht, eine komplett eigene Methode für dieses Problem zu programmieren. Deswegen wurde nach einer anderen Lösung gesucht. Anforderungen dabei waren, dass die Software sich einfach von Python aus bedienen lässt, sowohl unter Windows als auch unter Linux ohne großen Aufwand installierbar und benutzbar ist und eine freie Lizenz besitzt. Nach einigen Recherchen stellte sich heraus, dass das Python-Modul scipy die beste Wahl unter Python 3.2 ist.

Das Python-Modul scipy ist eine frei verfügbare Software für Mathematik, Naturwissenschaften und Technik und braucht zur Ausführung das Python-Modul

numpy (Numpy developers 2012). Alle Berechnungen auf Arrays basieren in scipy auf den Arrays aus numpy. Beide Module sind einfach zu bedienen, dennoch sehr umfangreich und mächtig (Scipy developers 2012). Aus der ganzen scipy-Bibliothek war für uns vor allem eine Klasse interessant, scipy.spatial.Delaunay. Die Klasse ermöglicht die Berechnung einer Triangulierung einer N-dimensionalen Punktwolke mittels des Delaunay-Algorithmus. Der Algorithmus, dessen Modifikation und Einsatz in der neuen Software, wird in Kapitel 2 (Seite 71) beschrieben. Jetzt steht die Installation der Module unter beiden Betriebssystemen im Vordergrund. Nach dem Herunterladen der Module (sie stehen auch in dem SVN-Verzeichnis zu der Anwendung zur Verfügung, siehe Anhang B, Seite 137) bleibt unter Windows nur noch, die entsprechenden Installer laufen zu lassen. Zuerst muss numpy installiert werden (Anhang C, Seite **Fehler! Textmarke nicht definiert.**). Danach kann man das scipy-Modul installieren (Anhang C, Seite **Fehler! Textmarke nicht definiert.Fehler! Textmarke nicht definiert.**). Die umgekehrte Reihenfolge führt zu Fehlern bei der Installation. Unter Linux wird die Installation nur dann erfolgreich enden, wenn alle Bibliotheken, die die Module, benutzen auch vorhanden sind. Die Reihenfolge der Installation muss genauso wie unter Windows sein. Jedoch müssen unter Linux ganz am Anfang ein paar andere Bibliotheken und Softwarepakete installiert werden, die die wissenschaftlichen Tools zur eigenen Installation brauchen werden. Welche Pakete noch gebraucht werden, kann man der Ausgabe des Installationsbefehls entnehmen.

```
atlas_blas_info:
  libraries f77blas,cblas,atlas not found in /usr/local/lib
  libraries f77blas,cblas,atlas not found in /usr/lib/sse2
  libraries f77blas,cblas,atlas not found in /usr/lib
  NOT AVAILABLE
```

Aus der Zeilen geht hervor, dass numpy z.B. die Bibliotheken Atlas und Blas für die Installation braucht. Bei der Linux-Installation wurden folgende Paketen nachgerüstet:

- g++
- gfortran
- libatlas
- libblas
- liblapack

Alle Abhängigkeiten wurden mit dem Ubuntu Software Center nachinstalliert. Alternativ kann man das in der Konsole mit dem Befehl sudo apt-get install [Name des Paketes] machen. Für die Installation der Pakete libatlas, libblas und liblapack wurde jeweils eine Entwicklerversion gewählt. Nachdem der Prozess abgeschlossen war, wurden die numpy-Quellen entpackt und die Lokalisation richtig gesetzt. In der Konsole muss man sich in dem Installationsverzeichnis für numpy befinden.

```
cd /path/to/numpy-1.6.2
```

Danach kann man das Modul mit dem Befehl build übersetzen und anschließend mit install installieren. Dies kann mitunter, abhängig von der Geschwindigkeit des Rechners, ein paar Minuten dauern.

```
python3 setup.py build
sudo python3 setup.py install
```

Zum Schluss sollte man bei der Ausgabe des Installationsbefehls noch überprüfen, ob keine Softwarepakete gefehlt haben. Wenn alles in Ordnung ist, genauso wie bei der Installation des pyserial-Moduls, wird das Installationsverzeichnis verlassen und getestet. Der Test verläuft sowohl unter Linux als auch unter Windows gleich und ist erfolgreich, wenn das Laden in Python funktioniert.

```
kasia@speedy:~/Downloads/numpy-1.6.2$ cd ..
kasia@speedy:~/Downloads$ python3
Python 3.2.2 (default, Sep  5 2011, 22:09:30)
[GCC 4.6.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
```

```
C:\Users\Kasia>python
Python 3.2.2 (default, Sep  4 2011, 09:51:08) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import numpy
>>>
```

Die Installation von scipy unterscheidet sich von der numpy-Installation nicht. Sowohl der Installer für Windows als auch die Quellen für Linux sind im Internet verfügbar (Python Developers 2012). Die Pakete sind auch im SVN-Verzeichnis abgespeichert (Anhang B, Seite 137). Unter Windows startet man den Windows-Installer für scipy und lässt sich durch die Installationsschritte führen (Anhang C, Seite **Fehler! Textmarke nicht definiert.**). Unter Linux führt man den build- und den install-Befehl in dem Installationsverzeichnis von scipy aus. Wenn zuerst numpy und andere Abhängigkeiten installiert wurden, verläuft der Prozess reibungslos und fehlerfrei. Nach einer erfolgreich abgeschlossenen Installation bleibt dann nur noch zu testen, ob das Modul sich ohne Fehlermeldung in Python mit dem Befehl import laden lässt.

Töne - winsound und pygame

Was noch gebraucht wird, ist eine Möglichkeit, Töne abzuspielen. Es ist sinnvoll, im Laufe der Vermessung einem Benutzer, der keinen Bildschirm sehen kann, akustische Signale schicken zu können. Mit den Signalen kann er schnell feststellen, in welchem Zustand sich die Anwendung befindet. Das hilft ihm, richtige Entscheidungen ohne weiteren Zeitverlust zu treffen, und beschleunigt somit den Arbeitsverlauf.

Unter Windows ist in der Python-Installation ein geeignetes Modul, winsound, schon enthalten. Es wird einfach mit dem Befehl import in der Python-Konsole geladen und ist fertig zur Benutzung. Mit winsound wird unter Windows 7 nur das

beep-Signal, das verschiedene Frequenzen und Längen hat, benutzt und keine selbst erstellten Signale, wie das unter Linux der Fall ist.

```
winsound.Beep(200, 150)
```

Der Grund dafür ist, dass sich in der Entwicklung herausgestellt hat, dass die Treiber für das Abspielen der Töne unter Windows XP und Windows 7 leicht verschieden sind und das Abspielen der wav-Dateien mit dem winsound-Modul nicht immer zuverlässig unter Windows 7 läuft. Unter Windows XP werden die gleichen Töne, die für Linux erstellt wurden, benutzt. Hier ist das Abspielen zuverlässig und stört den Programmablauf nicht. Außerdem ist der Klang des Beep-Tons unter Windows XP alles andere als angenehm. Um eine wav-Datei mit winsound in Python abzuspielen, wird der Befehl winsound.PlaySound ausgeführt.

```
winsound.PlaySound("C:\\...\\path\\to\\sound.wav", SND_FILENAME)
```

Unter Ubuntu 11.10 ist die Möglichkeit des Abspielens eines einfachen beep-Tons aus der Konsole heraus unterdrückt. Deshalb wurden Recherchen und Tests durchgeführt, die die zur Verfügung stehenden Möglichkeiten verifiziert haben. Es wurde entschieden, die Python-Medienbibliothek pygame zu benutzen. Dies führt zwar zu einem Overhead, weil das Modul für Spielentwicklung gedacht und deswegen sehr umfangreich ist. Für das Abspielen von ein paar einfachen Tönen in der Anwendung war das aber die bessere Alternative als z.B. das Kommandozeilentool aplay und ähnliche Lösungen, die, wie die Proben gezeigt haben, den Programmablauf sehr verlangsamt haben. Die Bibliothek pygame ist in Python geschrieben und erfordert, dass die SDL-Multimedia-Bibliothek so wie ein paar andere Software-Pakete auf dem Rechner vorhanden sind (Simple DirectMedia Layer Developers 2012). Der größte Aufwand bei der Installation von pygame unter Linux ist, die Abhängigkeiten zu finden und sie dann in dem System zu installieren (Pygame Developers 2012). Erst danach kann man mit der eigentlichen pygame-Installation beginnen.

Der erste Schritt ist das Herunterladen der entsprechenden Quellen. Dafür wird das Programm mercurial benutzt, da die richtigen Quellen für Python 3.2 auf der Internetseite von pygame leider nicht richtig verlinkt sind. Man kann sie aber in einem Archiv finden.

```
sudo apt-get install mercurial
hg clone https://bitbucket.org/pygame/pygame
```

Die herunterladenden Quellen wurden auch in dem Anwendungsverzeichnis abgespeichert (Anhang B, Seite 137). Nach dem Herunterladen und Entpacken wechselt man in das pygame-Installationsverzeichnis, wo die Quellen gebaut werden.

```
cd path/to/pygame
python3 setup.py build
```

Während des Übersetzens werden in der Konsole Informationen darüber angezeigt, ob alle Bibliotheken, die pygame braucht, gefunden wurden. In einem neu aufgesetzten Ubuntu 11.10 müssten folgende fehlende Abhängigkeiten ergänzt werden:

- libsdl-mixer1.2-dev
- libjpeg62-dev
- libsdl-image1.2-dev
- libportmidi-dev
- libsdl-ttf2.0-dev
- libavformat-dev
- libswscale-dev

Nachdem alle fehlenden Bibliotheken korrekt installiert wurden, wurden die Quellen mit dem setup.py-Skript fehlerfrei gebaut.

```
python3 setup.py build
```

Bei der Installation hat sich herausgestellt, dass es für pygame sinnvoller war, die Bibliothek als binäres Paket zu installieren. Dies ist auch eine Empfehlung der Entwickler der Bibliothek (Pygame Developers 2012). Es hat den zusätzlichen Vorteil, dass bei Bedarf das Modul einfach mit dem Standardpaketmanager entfernt werden kann (dpkg -r pygame). Zur Erstellung des binären Paketes wurde zuerst der checkinstall Tracker benutzt. Er verfolgt alle Änderungen, die ein Installationsskript in einem System durchführt, und baut ein binäres Debian-Paket für die Standardpaketverwaltung. Nach der Installation des Paketmanagers aptitude wird der Tracker installiert und anschließend ein binäres Paket erstellt.

```
sudo apt-get install aptitude
sudo aptitude install checkinstall
sudo checkinstall python3 setup.py install
```

Während der Erstellung fragt das Programm nach einem Namen für das Paket. Da der Name pygame gewählt wurde, muss die Installation des Moduls wie folgt durchgeführt werden:

```
sudo apt-get install pygame
```

Wie in anderen Fällen, wird auch hier die Installation zum Schluss in der Python-Konsole getestet. Falls keine Fehlermeldung erscheint, war die Installation erfolgreich und man kann die Töne abspielen. Dabei werden die wav-Dateien mit Hilfe des Objektes pymix verwaltet. Wichtig ist das Initialisieren des Objektes,

bevor man die Dateien lädt und abspielen lässt. Dabei werden alle Parameter für das Abspielen der Musik gesetzt.

```
import pygame.mixer as pymix
pymix.init()
pymix.music.load(path/to/sound.wav)
pymix.music.play(0)
```

Bei der Benutzung der Tondateien in Blender muss man darauf achten, dass in der Erweiterung die Pfade nicht absolut angegeben werden. Das Addon soll schließlich auf verschiedenen Maschinen laufen können, ohne dass man etwas an dem Quellcode ändern muss. Auch dafür liefert Blender entsprechende Operatoren. Im nächsten Kapitel (Seite 57) wird ihre Benutzung näher erklärt. Das Modul, das die Töne abspielen kann, ist jetzt vorhanden. Allerdings sind noch keine Töne für die Erweiterung erstellt worden. Die Töne wurden auf der Grundlage der Töne aus der Linux-Distribution mit Hilfe des Programms audacity erstellt (Abb. 15). Die Grundlage für die Erstellung der Signale wurde aus dem Verzeichnis /usr/share/sounds/ubuntu/stereo entnommen. Die neuen Dateien wurden in dem gleichen Verzeichnis, in dem sich der Quellcode der Anwendung befindet, abgespeichert. Auf diese Weise kann man von Blender aus die Pfade zu den Tönen relativ bauen lassen.

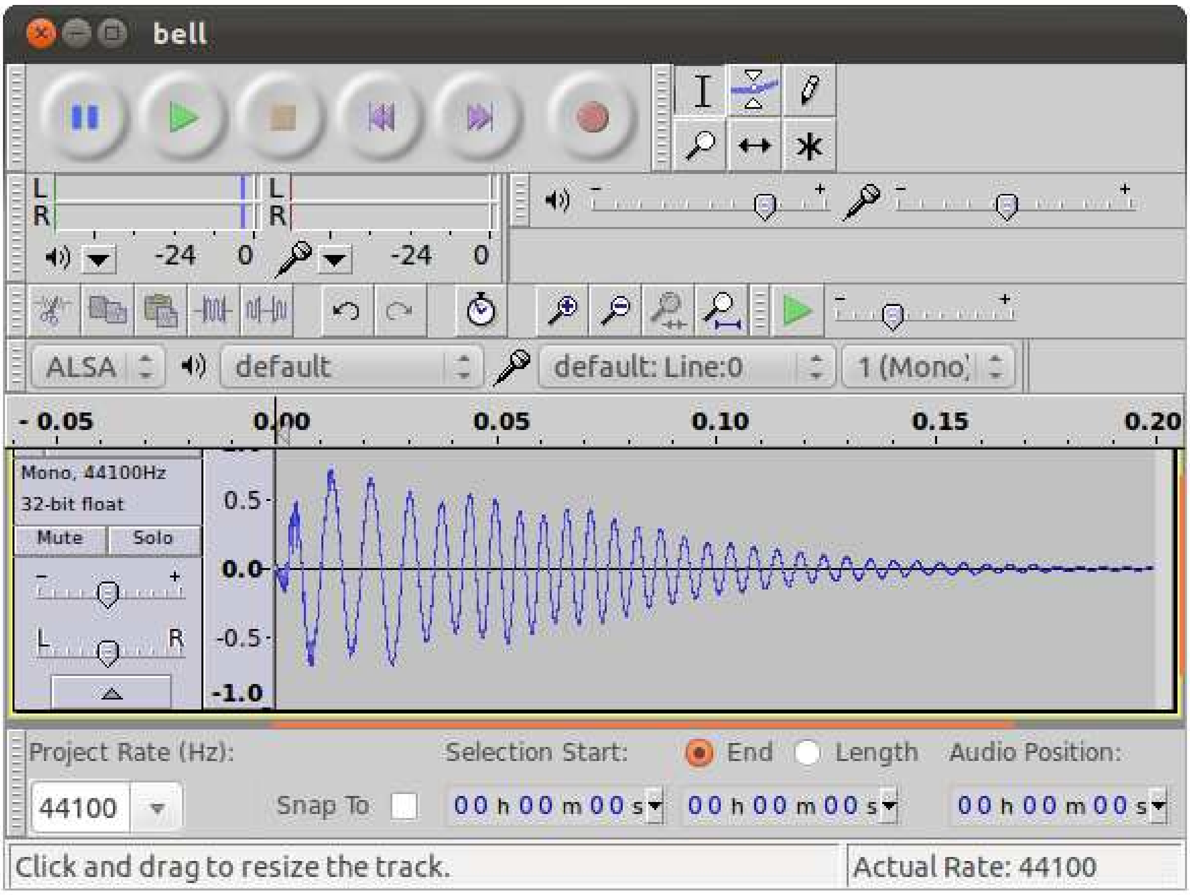


Abb. 15: Audacity **- ein Editor für Audiodateien.**

Einbinden von externen Python-Modulen in Blender

Alle benötigten externen Module sind zurzeit im Verzeichnis für die externen Module der systemweiten Python-Installation enthalten. Solange Blender den Pfad nicht kennt, können die Module von Blender aus nicht benutzt werden. Die Lösung ist entweder, in jedem Skript, das die externen Module braucht und in Blender integriert werden soll, den Systempfad von Blender zu ergänzen, oder man kopiert die erstellten Erweiterungen in das Blender-Python-Unterverzeichnis, so dass Blender darauf zugreifen kann. Danach kann man das Verzeichnis mit Blender und den Erweiterungen weitergeben, ohne dass man zusätzlichen Aufwand für die Installation betreiben muss. Das richtige Verzeichnis dafür hat unter Windows und Blender 2.62 den Pfad:

```
'C:\path_to_istallation_directory\Blender
```

Hier werden die Module pyserial, numpy und scipy kopiert. Unter Windows befinden sie sich im site-packages-Verzeichnis der Python-Installation.

```
'C:\Python32\Lib\site-packages'
```

Nachdem die Module sich in den entsprechenden Verzeichnissen befinden, kann man die Anwendung starten und in der Python-Konsole von Blender testen, ob der Interpreter die Module laden kann.

```
PYTHON INTERACTIVE CONSOLE 3.2 (r32:88445, Mar  4 2011, 02:13:51) [MSC v.1500 32 bit (Intel)]

Command History:     Up/Down Arrow
Cursor:              Left/Right Home/End
Remove:              Backspace/Delete
Execute:             Enter
Autocomplete:        Ctrl+Space
Ctrl +/-  Wheel:     Zoom
Builtin Modules:     bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, mathutils
Convenience Imports: from mathutils import *; from math import *

>>> import serial
>>> import numpy
>>> import scipy
>>>
```

Das gleiche wird unter Linux durchgeführt, mit dem Unterschied, dass hier das Verzeichnis mit zusätzlichen Python-Modulen dist-packages und nicht site-packages heißt und zusätzlich das Modul pygame kopiert wird.

```
'/usr/local/lib/python3.2/dist-packages'
```

Die Module werden aus dist-packages in das Zielverzeichnis der Blender-Installation kopiert.

```
'/path/to/installation_directory/.../2.62/python/lib/python3.2'
```

Auch hier sollte man in Blender testen, ob der Interpreter die neuen Pakete laden kann und keine Fehlermeldungen erscheinen.

Kapitel 2

Implementierung der Scan-Erweiterung

In diesem Kapitel wird die Struktur des Quelltexts für die neue Erweiterung von Blender beschrieben. Es werden sowohl die mathematischen Hintergründe der Implementierung, als auch die wichtigsten programmiertechnischen Probleme erklärt. Die Beschreibung der Implementierung folgt dem Aufbau des Addons. Bei jedem Quelltextteil wird, bei Bedarf, außer der Technik auch die Theorie, die für das Programmieren gebraucht wurde, erklärt. Die Quelltextstücke werden übersichthalber kurz gehalten und oft nur als Pseudocode angegeben. Der vollständige Code des Addons befindet sich in dem Verzeichnis 3D_SCAN_ANWENDUNG des SVN-Archivs (Anhang B, Seite 137).

Struktur der Erweiterung

Das Prototypprogramm aus dem Praktikum erlaubt eine gute Abschätzung dessen, was die Hauptteile der Erweiterung beinhalten sollen. Die Hauptbestandteile der Implementierung sind in Abb. 16 dargestellt. In wesentlichen waren die Schnittstelle zum Scanner und zum Benutzer zu programmieren, die Darstellung und Organisation der gescannten Daten und die Schnittstelle zum Export der Daten in Formaten, die in anderen Programmen benutzt werden können.

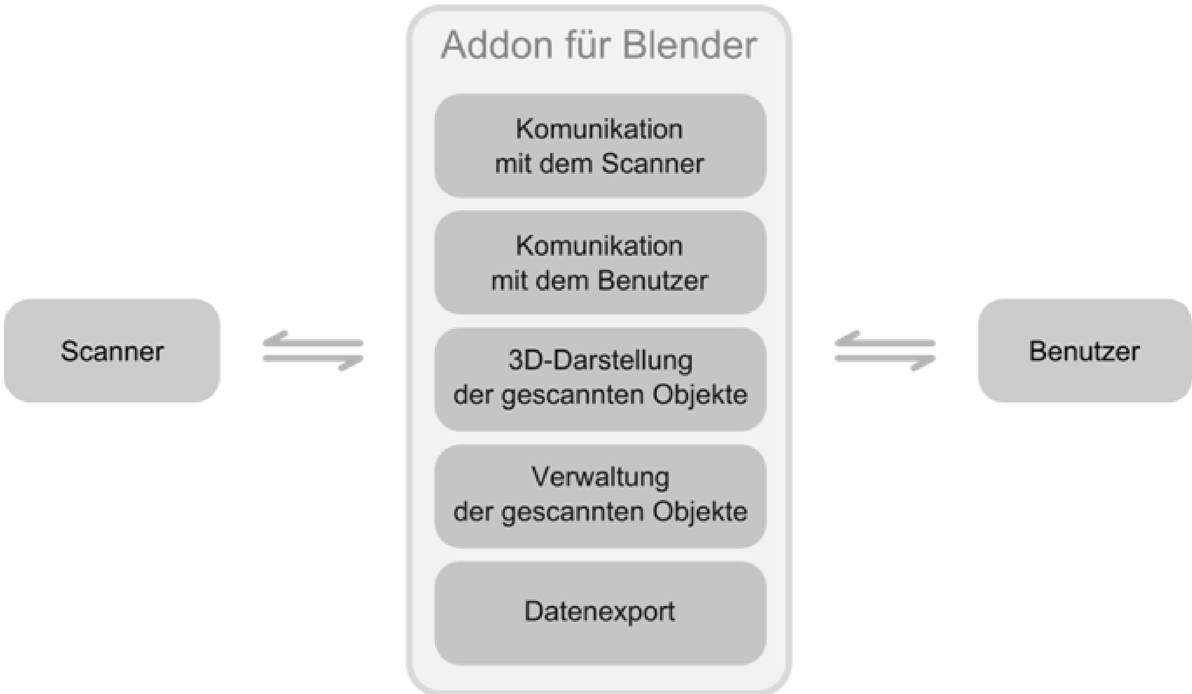


Abb. 16: Die implementierten Funktionalitäten der Blender-Erweiterung zum Scannen von Pflanzen mit Hilfe des Trackingsystems FASTRAK.

Der entworfene Quelltext wurde thematisch so aufgeteilt, dass die Struktur das Suchen nach den Fehlern, die Wartung und das Einsteigen in den Quellcode für einen neuen Programmier erleichtet. Das Addon-Verzeichnis befindet sich in dem Installationsverzeichnis für Blender-Erweiterungen und ist damit in Blender integriert, so dass man das Addon von der Oberfläche in Blender starten kann (Abb. 39).

```
# Windows:
'...\path\to\installation_directory\Blender\2.62\scripts\addons\fkscn'

# Linux:
'.../path/to/installation_directory/.../2.62/scripts/addons/fkscn'
```

Die Erweiterung befindet sich auch im SVN-Verzeichnis der Anwendung (Anhang B, Seite 137). Im Addon-Verzeichnis befinden sich außer dem Quelltext auch die Töne, die für die Vermessung unter Ubuntu 11.10 und Windows XP vorbereitet wurden. Der Quellcode ist auf sieben Dateien, bzw. Python-Module, aufgeteilt. In jedem Modul befinden sich Klassen und/oder Funktionen, die thematisch zusammen gehören. In dem fkscn-Addon befinden sich dementsprechend folgende Python-Dateien:

- __init__.py
- f_controller.py
- f_labels.py
- f_obj_factory.py
- f_operators.py
- f_properties.py
- f_ui.py

Darüber hinaus sind in dem Addon-Verzeichnis die wav- und ogg-Dateien, die für die Windows XP und Linux-Version des Addons erstellt wurden, abgespeichert. Die Erweiterung verfügt über acht verschiedene Signale für den Scanvorgang:

- add_group.wav (add_group.ogg)
- connect_off.wav (connect_off.ogg)
- connect_on.wav (connect_on.ogg)
- error.wav (error.ogg)
- finish_draw.wav (finish_draw.ogg)
- first_point.wav (first_point.ogg)
- last_point.wav (last_point.ogg)
- pop_group.wav (pop_group.ogg)

Das Modul __init__.py

Das __init__.py-Modul beinhaltet die register- und unregister-Methoden von Blender. Die wichtigste Klasse des Moduls ist SCENE_OT_fk_scan_manager. Sie steuert das Zeichnen der vom Scanner empfangenen Daten in Blender. Desweiteren befinden sich hier Funktionen und Blender-Operatoren, die die Verwaltung der seriellen Verbindung unterstützen. Es werden in den nächsten Abschnitten die wichtigsten Details zu den Klassen und Funktionen aus diesem Modul besprochen.

Steuerung und Verwaltung der Eingabedaten

Die Klasse SCENE_OT_fk_scan_manager für das Management der Darstellung der ankommenden Daten ist auch ein Blender-Operator, wie in Kapitel 1 (Seite 27) dargestellt, besitzt aber eine zusätzliche Funktion modal(self, context, event). Dank dieser Funktion ist eine interaktive Benutzung des Programms ohne spürbare Verspätungen möglich. Somit kann man während der Datenaufnahme die Oberfläche von Blender normal benutzen. Der Operator wird in definierten Zeitabständen solange aufgerufen und ausgeführt, bis die Abbruchbedingung in der modal-Methode erfüllt ist. Nur dann wird der gewöhnliche Rückgabewert {'FINISHED'} zurückgegeben. Andere Rückgabewerte, die beim Operator auftreten können, sind {'RUNNING_MODAL'} und {'PASS_TROUGH'}. Beim ersten Aufruf des Operators wird die invoke-Methode aufgerufen. In der Methode wird zuerst getestet, ob alle Bedingungen für das Öffnen der seriellen Verbindung erfüllt sind. Ist das der Fall, wird am Ende der Rückgabewert {'RUNNING_MODAL'} ausgegeben und die Ausführung geht zu den Funktionen execute und modal über. Der Rückgabewert {'RUNNING_MODAL'} ist ein Zeichen dafür, dass der Operator nach einer kurzen Pause wieder aufgerufen wird.

```
def invoke(self, context, event):

    ## if connection successfully open
    return {'RUNNING_MODAL'}

    ## else connection failed
    return {'FINISHED'}
```

In der execute-Methode befindet sich der Teil des Quelltextes, der für die wiederholten Aufrufe des Operators sorgt. Es wird hier dem Blender-Fenster-Manager, der für die Verwaltung der Elemente der Oberfläche zuständig ist, signalisiert, dass er ein neues timer-Event zu verwalten hat. Wie bei der invoke-Methode ist der Rückgabewert in dem Fall {'RUNNING_MODAL'}.

```
def execute(self, context):
    manager = context.window_manager
    manager.modal_handler_add(self)
    self._timer = manager.event_timer_add(0.001, context.window)
    return {'RUNNING_MODAL'}
```

Die dritte wichtige Funktion des Operators, die für die eigentliche, modale Ausführung entscheidend ist, ist die Methode modal. Hier befinden sich die Abbruchbedingung und die Aufrufe der Funktionen, die die Daten von dem Scanner anfordern und lesen, sie danach für die Darstellung vorbereiten, die Position des

Cursors setzen und schließlich die Objekte erstellen. Wenn die Methode abgearbeitet wurde, liefert sie den Rückgabewert {'PASS_TROUGH'}. Somit bleibt der Operator immer noch aktiv. Er wird in kurzen Abständen immer wieder mit der Methode modal gestartet. Wie man der Blender-Dokumentation (Blender Foundation 2012) entnimmt, ist das die beste Art und Weise, wie man in Blender dem Benutzer die Oberfläche frei gibt, obwohl gleichzeitig ein Skript ausgeführt werden soll.

```
def modal(self, context, event):
    scn = context.scene

    ## Finish
    if not scn.fk_contr.started:
        # close connection
        return {'FINISHED'}

    ## Draw
    if event.type == 'TIMER':
        # read and prepare scan data
        # set cursor position
        # draw curent shape
        return {'PASS_THROUGH'}
```

Die Benutzung des threading-Modules wird hier nicht empfohlen, da eine sichere Benutzung von Python-Threads in Blender zurzeit nicht unterstützt wird. Man muss dabei beachten, dass der Teil, der in der Methode modal ausgeführt wird, nicht zu rechenintensiv ist. Anderenfalls würde der Benutzer die Wartezeiten doch wahrnehmen. Wenn ein TIMER-Event aufgetreten ist, werden die ankommenden Daten nach dem Lesen und vor dem Zeichnen überarbeitet. Der Grund dafür ist zum einen, dass die Daten auf der seriellen Verbindung nicht immer sauber übertragen werden, und zum anderen, dass bei Bedarf die Korrekturen für das Kalibrieren und die Verschiebung des Senders einsetzbar sein sollen. Die am häufigsten auftretenden Fehler bei der Aufnahme sind z.B. fehlendes Leerzeichen zwischen aufeinander folgenden Zahlen oder abgeschnittene, bzw. zu lange Datenzeilen. Jeder empfangene Datensatz muss immer aus drei Koordinaten, drei Rotationen, dem Status der Taste auf dem Stift und dem Umbruchzeichen bestehen. Dies muss entweder vor der Umwandlung der ASCII-Zeichen in Fließkommazahlen oder während der Umwandlung kontrolliert und abgefangen werden. Andernfalls wird die Ausführung des Programms angehalten. In der Klasse SCENE_OT_fk_scan_manager wird dies in der Methode __clear_line behandelt. Sie sorgt dafür, dass das Leerzeichen zwischen einer Zahl und einer Zahl mit einem negativen Vorzeichen immer vorhanden ist. Bei der Bereinigung wird die Eingabe in eine Liste von Strings umgewandelt und zurückgeben.

```
def __verify_line(self, raw_line):
    return str(raw_line).replace("-", " -").split()
```

Danach wird getestet, ob die Länge der Liste gleich acht ist, und falls ja, wird die Methode __prepare_float_line aufgerufen. Die Methode wandelt die Strings aus der Liste in Floats, die für Aufrufe der Blender-Operatoren benötigt werden, um. Falls nötig wird hier die Korrektur aus der Kalibierung und/oder Entzerrung auf

den Koordinaten des Punktes durchgeführt (siehe Seite 88). Die try/except-Klausel sorgt hier dafür, dass eventuell nicht abgefangene Fehler die ganze Ausführung nicht abbrechen. Ein Datensatz, der einen Fehler verursacht hat, wird einfach während des Ablaufs der modal-Methode ignoriert.

```
def __prepare_float_line(self, raw_line):
    try:
        # cast to float

        # data correction if needed

    except:
        return None
    return float_line
```

Für eine bessere Übersicht während des Scannens sollte es möglich sein, die Position des Stifts nachverfolgen zu können. Hierfür wurde die Möglichkeit des Setzens der neuen Position für den 3D-Cursor ausgenutzt. Das Update-Verhalten des Cursors kann man in der Anwendung über die Benutzeroberfläche steuern. Der Benutzer soll entscheiden können, nach wie vielen angekommenen Zeilen die Position des Cursors geändert werden soll. Die Methode für das Neuzeichnen der Cursorposition heißt __update_cursor. Sie wird in der modal-Methode der Klasse SCENE_OT_fk_scan_manager aufgerufen. Dabei werden zwei Hilfsvariablen, __lineCounter und __cursorHistory, benutzt. Die Variable __lineCounter zählt die ankommenden Datenzeilen bis zum nächsten Cursor-Update. Die Variable __cursorHistory enthält die aktuelle Anzahl der gewünschten Updates. Somit kann man über den Vergleich mit der Blender-Eigenschaft scene.fk_contr.cursor_updates sofort erkennen, wann der Benutzer den Update-Wert geändert hat. In jedem Ablauf der Funktion __update_cursor ändert sich der Zähler für Cursor-Updates. Wenn die Variable __cursorHistory einen anderen Wert als die Eigenschaft scene.fk_contr.cursor_updates besitzt, wird der Zähler auf Null gesetzt und im nächsten Aufruf mit der Eigenschaft für Cursor-Updates neu verglichen, inkrementiert oder wieder auf null gesetzt. Die neue Position des Cursors in der Szene wird mit der Zuweisung scene.cursor_location = location gesetzt.

```
__lineCounter = 0
__coursorHistory = 0

def __update_cursor(self, location, scene):

    # location update
    if self.__lineCounter == scene.fk_contr.cursor_updates:
        scene.cursor_location = location
        self.__lineCounter  = -1

    # counter management
    if self.__coursorHistory != scene.fk_contr.cursor_updates:
        self.__coursorHistory = scene.fk_contr.cursor_updates
        self.__lineCounter = 0
    else:
        self.__lineCounter = self.__lineCounter + 1
```

Desweiteren braucht man zum Verwalten des Scannens die Möglichkeit, den Status der Stifttaste zu erkennen. Dafür muss der Stift zuerst in den Mausmodus (siehe Seite 57) versetzt werden. Dabei muss das Ausgabeformat auf dem Scanner

entsprechend eingestellt werden (Tab. 3). Ab dem Zeitpunkt, an dem die Taste von einem Benutzer gedrückt wurde, wird in der Datenzeile eine Eins gesendet, und sonst eine Null. Auf diese Art und Weise kann die Vermessung sowohl über die Benutzeroberflächenelemente als auch über den Zustand der Empfängertaste angesteuert werden. Der alte und der aktuelle Zustand der Taste wird dieses Mal nicht in einer Klassenvariablen protokolliert, sondern in der Blender-Eigenschaft scn.fk_contr.fButton_state. So ist er aus allen Addon-Modulen greifbar. Der Grund dafür ist, dass verschiedene Funktionen und Operatoren im ganzen Addon die Information über die Veränderung des Zustandes der Empfängertaste brauchen, und nicht immer lässt sich die Information über die Übergabe des Wertes als ein Funktionsargument in der SCENE_OT_fk_scan_manager-Klasse realisieren. Nach dem Drücken der Taste befindet sich in der ankommenden Zeile am Ende eine 1. In der bearbeiteten Liste der Strings steht dann der Eintrag 1\r\n'. Dies entspricht den True und False Werten der Eigenschaft fButton_state. Die Aktualisierung des Zustands erfolgt für jede empfangene Datenzeile. Die verschiedenen Kombinationen aus den Werten [True, False] werden für die Steuerung des kontinuierlichen Zeichnens der Objekte genutzt. Die Beschreibung der Bedeutung der Belegung der fButton_state-Variable während des Zeichnens der Objekte wird im weiteren Verlauf des Kapitels bei der Besprechung des Zeichnens der Objekte angegeben (siehe Seite 73).

```
def __update_fButtonState(self, scn, line_state):

    # receiver button state history

    if line_state == r"1\r\n'":
        scn.fk_contr.fButton_state = [scn.fk_contr.fButton_state[1], True]

    else:
        scn.fk_contr.fButton_state = [scn.fk_contr.fButton_state[1], False]
```

Verwaltung des Objekts zum Steuern der Kommunikation

Die Klasse SCENE_OT_fk_scan_manager besitzt selbst keine Funktionen, die den Scanner oder das Objekt für die serielle Kommunikation direkt ansteuern. Die erste Funktion des Addons, die diesem Zweck dient, ist init_con. Sie wird bei der Registrierung, in der invoke-Methode der Klasse SCENE_OT_fk_scan_manager und in dem Operator SCENE_OT_fk_con_reload, der die Verbindung mit dem Scanner erneut erstellt, aufgerufen. In der Funktion wird das globale serielle Objekt ser der Klasse FASTRAK_Controller (siehe Seite 57), falls nötig, neu erstellt oder vernichtet. Es ist sehr wichtig, dafür zu sorgen, dass in dem ganzen Addon nur ein serielles Objekt existiert und nur dieses Objekt exklusiv über die serielle Schnittstelle mit dem Scanner kommuniziert. Wenn die Schnittstelle schon besetzt ist, z. B. durch eine andere Anwendung, schlägt der Versuch, sie anzusprechen, fehl. Entsprechend der Situation werden die Blender-Eigenschaften für den Scan ser_obj_ok und started auf True oder False gesetzt. Die Klassen und Operatoren in der ganzen Erweiterung können somit in ihrer poll-Methode die Gültigkeitsbereiche entsprechend setzen.

```
def init_con(context):
    global ser
    ok = False
    try:
        if not ser:
            ser = FASTRAK_Controller()
            ok = True
            context.scene.fk_contr.ser_obj_ok = 1
    except:
        context.scene.fk_contr.ser_obj_ok = 0
        context.scene.fk_contr.started = 0
        ser = None
    return ok
```

Kontinuierliches und punktweises Senden der Daten

Der Scanner beherrscht zwei Modi, Daten über die serielle Schnittstelle zu senden. Im kontinuierlichen Modus werden die Daten ununterbrochen auf die Schnittstelle geschickt. Wenn nur ein Empfänger angeschlossen ist, beträgt die Geschwindigkeit 120 Updates pro Sekunde. Im Einzelpunktmodus wird mit dem Verschicken der Daten gewartet, bis sie von der Softwareseite oder über die Empfängertaste angefordert werden. Bei verschiedenen Messungen werden verschiedene Verfahren bevorzugt. In der Erweiterung wird der Einzelpunktmodus während der Aufnahme der Punkte für die Kalibrierung, Korrektur der Lage nach dem Verschieben des Senders, Setzen der geographischen Richtung und Standardabmessung der Topologie der Pflanzen eingesetzt. In allen anderen Fällen wird der kontinuierliche Modus genutzt. Der Operator, der diese Eigenschaft des Scanners umschaltet, heißt SCENE_OT_fk_switch_scan_mode. Er wird in der invoke-Methode der Klasse SCENE_OT_fk_scan_manager und in den Updatefunktionen der Eigenschaften (Tab. 6), die für das Umschalten zwischen verschiedenen Panels für die Vermessung zuständig sind, aufgerufen. Das Umschalten selbst wird mit Hilfe des ser-Objekts durchgeführt. Wie die einzelnen Befehle genau aussehen und welchen Zwecken sie dienen, wird in weiteren Abschnitten des Kapitels erklärt (siehe Seite 57). Es ist wichtig zu wissen, dass die Reihenfolge der Befehle einen Einfluss auf den Erfolg bei der Umschaltung hat. Wenn der Scanner im Einzelpunktmodus arbeiten und die Empfängertaste funktionieren soll, muss man zuerst die Befehle ser.set_butt_point_mode und ser.set_zero_incr an die serielle Schnittstelle senden. Erst dann kann der Befehl ser.set_point_mode für die Umschaltung geschickt werden. Bei der Umschaltung in den kontinuierlichen Modus muss man den Stift zuerst wieder in den Mausmodus versetzen, um das Verhalten des Scanners über die Software zu steuern. Sonst wird das Drücken der Stifttaste das Senden der Daten im kontinuierlichen Modus anhalten und starten. Nach dem Befehl ser.set_butt_mouse_mode hat das Drücken der Taste keinen Einfluss mehr auf das Senden der Daten. Für den kontinuierlichen Modus kann mit dem Befehl ser.set_other_incr_value auf dem Scanner ein Mindestabstand zwischen zwei zu sendenden Punkten eingestellt werden. Das reduziert die Menge der an der seriellen Schnittstelle ankommenden Daten und beschleunigt dadurch das Zeichnen der Objekte. Bevor man den Scanner wieder in den Einzelpunktmodus umschaltet, muss der Inkrementwert wieder auf Null zurückgesetzt werden. Falls das Inkrement auf dem Scanner von Null verschieden

ist, wird nach dem Umschalten in den Punktmodus kein Datensatz gesendet. Am Ende wird der Befehl für das Setzen des kontinuierlichen Modus aufgerufen.

```
class SCENE_OT_fk_switch_scan_mode(bpy.types.Operator):
    # Class body...
    def execute(self, context):
        # if calibration, references, geo direction, standard topology:
            # switch to POINT MODE
            ser.con.write(ser.set_butt_point_mode)
            ser.con.write(ser.set_zero_incr)
            ser.con.write(ser.set_point_mode)
        # else:
            # switch to CONTINUOUS MODE
            ser.con.write(ser.set_butt_mouse_mode)
            ser.con.write(ser.other_incr_value(scn.fk_contr.increment))
            ser.con.write(ser.set_cont_mode)

        return {'FINISHED'}
```

Integration in Blender

Das Modul __init.py enthält auch die Blender-Funktionen register und unregister, die für die Integrationsverwaltung verantwortlich sind. In der Funktion register befinden sich deshalb Aufrufe der Operatoren bpy.utils.register_class und bpy.utils.register_module, die das Anmelden ermöglichen. Der erste Operator wird für den modalen Operator SCENE_OT_fk_scan_manager verwendet. Dies ermöglicht z.B. das Schließen der seriellen Verbindung und das Setzen der Eigenschaften für das serielle Objekt ser sogar dann, wenn die Verbindung nicht über die Checkbox auf der Oberfläche, sondern z. B. über die unregister Funktion abgebrochen wurde. Alle anderen Blender-Operatoren können mit bpy.utils.register_module eingebunden werden. In der Funktion register werden auch alle Eigenschaften für den Scan Addon initialisiert und der Szene oder dem Objekt zugeordnet. Es werden auch die Informationen, ob in dem Programm schon die Objekte für farbige Markierung des Modells, und die Scangruppen angelegt sind, aktualisiert. Falls nötig, werden die fehlenden Teile ergänzt (siehe Seite 60). In dieser Methode wird in der Linux-Version des Addons auch das pygame-Modul mixer initialisiert. Die unregister-Methode macht die Änderungen ruckgängig.

```
def register():
    # init pygame module for linux
    # ...

    # register class and modules
    # ...

    # initialise the properties for scene and for object
    # ...

    # change settings and properties if the dataset already exists
    # ...

    # check the connection and set up the serial object


def unregister():
    # revert the changes
    # ...


# starts the addon
if __name__ == "__main__":
    register()
```

Das Modul f_controller.py

In dem Modul f_controller.py befindet sich die Klasse FASTRAK_Controller, die sehr wichtig für die Verwaltung der seriellen Verbindung ist. Sie kapselt die Befehle für den Scanner ab und sucht bei der Initialisierung der Verbindung nach dem Gerät. Desweiteren ist in dem Modul die Funktion für das Abspielen der Töne untergebracht. Alle anderen hier definierten Funktionen helfen beim Darstellen und Verwalten des gescannten Modells.

Steuerung des Scanners

Für die Vermeidung von Fehlern bei mehrfachen Definitionen werden die Steuerbefehle für den Scanner alle in der Klasse FASTRAK_Controller des f_controller.py-Moduls gesammelt. Die Bedeutung der einzelnen Zeichen oder der Zeichenzusammensetzung der Befehle wurde in Tab. 3 in Kapitel 1 besprochen. Deshalb wird hier nur auf die Implementierung eingegangen.

Der Attribut get_state kapselt den Befehl für die Statusabfrage ab. Die Befehle, die auf den seriellen Port mit der Funktion write geschickt werden, müssen jeweils in Bytes umgewandelt werden.

```
get_state = bytes('S', 'latin1')
```

Wenn der Scanner im Einzelpunktmodus arbeitet, wird der Befehl get_point für das Anfordern des Punktes bei der Datenaufnahme gebraucht. Dieser Befehl funktioniert nur dann korrekt, wenn die Entfernung der Punkte zur Ausgabe im Einzelpunktmodus auf Null gesetzt ist.

```
get_point = bytes('P', 'latin1')
```

Der Benutzer kann in der Oberfläche die Entfernung für das Inkrement des Geräts beliebig ändern. Deshalb werden in dem Quelltext die zwei nächsten Befehle set_zero_incr und other_incr_value in der Klasse FASTRAK_Controller definiert.

```
set_zero_incr = bytes('I1,0.0\x00\r\n', 'latin1')
def other_incr_value(self, increment):
    return bytes('I1,%s\x00\r\n'%increment, 'latin1')
```

Sowohl in der kontinuierlichen als auch in der punktweisen Datenaufnahme wird in der Erweiterung ein bestimmtes Format der Datenzeile gebraucht. Für das Steuern der Vermessung ist es vor allem wichtig, dass man erkennen kann, ob der Benutzer die Taste auf dem Stift gerade gedrückt hat. Mit dem Befehl set_line_format wird festgelegt, dass in der Ausgabe immer die drei Koordinaten der Translation, die drei Rotationen des Stiftes und der Status der Stifttaste gesendet werden.

```
set_line_format = bytes('O1,2,4,16,1\x00\r\n', 'latin1')
```

Der Stift kann in zwei verschiedene Modi gesetzt werden. Der Verlauf der Vermessung bei der kontinuierlichen Aufnahme der Daten wird mit der Stifttaste im Mausmodus gesteuert. Zur Aufnahme der Daten im Einzelpunktmodus muss auch für die Stifttaste der Einzelpunktmodus (eng. Trackermode) statt des Mausmodus eingestellt sein. Wenn der Stift mit set_butt_mouse_mode in den Mausmodus versetzt wird, hat die Taste keinen Einfluss auf das Senden der Daten. Im kontinuierlichen Modus ändert sich dann nur der Wert für den Status der Taste. Im Einzelpunktmodus passiert nichts, solange die Daten aus der Software nicht angefordert werden. Wenn der Stift mit set_butt_point_mode in den Einzelpunktmodus versetzt wurde, kann man die Aufnahme der Daten direkt über die Taste steuern. Dabei wird im kontinuierlichen Modus des Scans das Senden nach dem Drücken der Taste angehalten oder wieder gestartet. Im Einzelpunktmodus hat das Drücken der Taste die gleiche Wirkung wie das Senden des Befehls get_point.

```
set_butt_mouse_mode = bytes('e1,0\x00\r\n', 'latin1')
set_butt_point_mode = bytes('e1,1\x00\r\n', 'latin1')
```

Das Umschalten zwischen den zwei Modi des Scanners wird über die Befehle set_point_mode (Einzelpunktmodus) und set_cont_mode (kontinuierlicher Modus) gesteuert.

```
set_point_mode = bytes('c', 'latin1')
set_cont_mode  = bytes('C', 'latin1')
```

Der letzte abgekapselte Befehl dient der Umschaltung der Einheiten der Koordinaten von Inches auf Zentimeter. Beim Ablesen solcher Werte wie Volumen, Masse oder Fläche muss man, falls nötig, die Blender-Einheiten mit einem entsprechenden Faktor multiplizieren, um die Einheiten anzupassen.

```
set_metric_units = bytes('u', 'latin1')
```

Wenn ein Objekt der Klasse FASTRAK_Controller initialisiert wird, wird zuerst eine Liste der Portnamen für das aktuelle Betriebssystem erstellt. Unter Windows fängt die Liste mit COM1 an und endet mit COM256. Unter Linux ist die Liste kürzer und enthält Einträge von /dev/ttyS0 bis /dev/ttyS31. Danach wird in der Funktion __get_con für jeden Port getestet, ob sich an dem Port das Gerät befindet. Zum Test wird die Abfrage des Status verwendet. Wenn der Scanner gefunden wurde, gibt die Funktion __get_con das Objekt der Klasse Serial aus dem Modul pyserial zurück. Andernfalls wird das None-Objekt zurückgeliefert. Entsprechend des Rückgabewerts wird die Blender-Eigenschaft ser_obj_ok auf wahr oder falsch gesetzt.

```
def __init__(self):
    # list of the ports for current operating system
    self.__ports = self.__get_com_ports()
    # initialize the serial object if FASTRAK is enabled
    self.con = self.__get_con()
    # set the blender property for the status of the serial object
    if self.con:
        bpy.context.scene.fk_contr.ser_obj_ok = 1
    else:
        bpy.context.scene.fk_contr.ser_obj_ok = 0
```

Ausgabe von Signaltönen

Die Töne in der Erweiterung werden unter Windows XP, Windows 7 und Linux jeweils auf andere Weise, aber mit der gleichen Funktion play_sound, abgespielt. Deshalb wird am Anfang des f_controller.py Moduls die Funktion uname aus dem Standard-Python-Modul platform importiert. Der Rückgabewert der Funktion ist ein Tupel mit Strings, die folgende Informationen über den Rechner und das Betriebssystem enthalten:

- Systemname
- Name des Rechners
- Release des Betriebssystems
- Version des Betriebssystems
- Art der Architektur
- Prozessorname

```
from platform import uname
platformInfo = uname()
```

Zur Unterscheidung, auf welchem System die Töne abgespielt werden sollen, brauchen wir die Information über den Namen des Betriebssystems und über das Release des Systems, falls das Addon unter Windows benutzt wird. Dank der Unterscheidung werden nur die in dem Betriebssystem benötigten Funktionen und Module geladen. Unter Windows XP wird aus dem Modul winsound die Funktion PlaySound und die Konstante SND_FILENAME geladen. Unter Windows 7 wird, da die Funktion PlaySound hier nicht zuverlässig arbeitet, die Funktion Beep benutzt. Unter Linux werden die Töne mit Hilfe der Bibliothek pygame und des Moduls mixer abgespielt.

```
if platformInfo[0] == 'Windows' and platformInfo[2] == 'XP':
    from winsound import PlaySound, SND_FILENAME

if platformInfo[0] == 'Windows' and platformInfo[2] == '7':
    from winsound import Beep

if platformInfo[0] == 'Linux':
    import pygame.mixer as pymix
```

Beim Aufruf der play_sound-Funktion des Addons ist eine ganze Zahl von 1 bis 8 als Argument nötig. Die Abbildung der Zahlen auf die abzuspielenden Töne ist in Tab. 4 dargestellt.

Tab. 4: Abbildung der ganzen Zahlen auf die abzuspielenden Töne.

Zahl	Windows XP	Windows 7	Linux
1	connect_on.wav	Beep(500, 200)	connect_on.ogg
2	connect_off.wav	Beep(500, 150) Beep(500, 150)	connect_off.ogg
3	add_group.wav	Beep(450, 150)	add_group.ogg
4	pop_group.wav	Beep(450, 100) Beep(450, 100)	pop_group.ogg
5	first_point.wav	Beep(550, 200)	first_point.ogg
6	last_point.wav	Beep(590, 200)	last_point.ogg
7	finish_draw.wav	Beep(550, 150) Beep(550, 150)	finish_draw.ogg
8	error.wav	Beep(600, 150)	error.ogg

Im Rumpf der play_sound-Funktion wird mit Hilfe der if-Klausel getestet, welche Sound-Datei abgespielt werden soll. Alle Dateien liegen in dem Addon-Verzeichnis, weshalb die Erstellung des Pfades zu der ausgewählten Tondatei kein Problem darstellt. Die Funktion bpy.utils.script_paths liefert sowohl den Pfad zu dem Skriptverzeichnis der Blender-Installation als auch zu dem Benutzerverzeichnis der aktuellen Session. Für die Erstellung des Pfades für die Töne wird nur die erste Information gebraucht. Der Pfad zu dem Addon-Verzeichnis, wo sich auch alle Töne befinden, wird spezifisch für das jeweilige Betriebssystem erstellt. Wie man mit dem Modul winsound oder pygame die Töne abspielen kann, wurde schon in Kapitel 1 (Seite 44) besprochen.

```
# Windows
addon_path  = "%s%s"%(bpy.utils.script_paths()[0], '\\addons\\fkscn')

# Linux
addon_path  = "%s%s"%(bpy.utils.script_paths()[0], '/addons/fkscn')
```

Initialisierung des Addons in Blender

Im Modul f_controller.py sind die vier Funktionen definiert, die beim Laden des Addons den Zustand der Anwendung an die aktuelle Situation anpassen. Somit ist es möglich, nach dem Abspeichern des gescannten Modells die Informationen aus dem Scan weiter zu benutzen, mit der Messung weiter fortzufahren oder das Modell zu bearbeiten. Die Funktionen heißen create_scan_materials, init_groups, set_initial_mtg_lists und set_mtg_enum. Sie werden nur einmal beim Start des Addons aufgerufen. Der Aufruf befindet sich im __init.py-Modul in der Funktion register.

```
# module __init.py
def register():
    f_controller.create_scan_materials()

    f_controller.init_groups()

    f_controller.set_initial_mtg_lists()

    f_controller.set_mtg_enum("branch")

    f_controller.set_mtg_enum("unit")

    f_controller.set_mtg_enum("plant")
```

Die erste Funktion aus der Gruppe, create_scan_materials, erstellt, falls nötig, Materialien für die Färbung der gescannten Objekte. Dank der Materialien kann der Benutzer auf den ersten Blick erkennen, was für ein Objekttyp gerade gezeichnet wurde. Die bei der Abmessung der Topologie der Pflanze entstehende Geometrie wird beispielweise eine andere Farbe haben, wenn ein Internodium gezeichnet wurde, und wieder eine andere, wenn es ein Blatt, eine Frucht oder eine Blüte ist. Damit das Material nicht mehrfach in Blender definiert wird, wird jedes Mal in der Funktion getestet, ob die zu definierende Materialstruktur schon in der Anwendung vorhanden ist. Falls kein solches Material existiert, gibt die find_mat-Funktion False zurück und das entsprechende Material kann mit der Funktion make_material erstellt werden. In der Tab. 5 sind alle in dem Addon benutzten Materialen sowie deren Zuordnung zu den gescannten Objekten zusammengestellt.

```
def create_scan_materials():

    if not find_mat(material_name):
        make_material(material_name, par_1, par_2)

    # similarly for other materials ...
```

Genauso wie im Falle der Materialien wird das Testen und Erstellen der Gruppen für die gescannten Objekte in der Funktion init_groups durchgeführt. Eine fehlende Gruppe wird dann erstellt, wenn in der Blender-Kollektion bpy.data.groups der Name einer Gruppe nicht gefunden und eine Ausnahme ausgeworfen wird. Wenn das passiert, werden mit der Funktion deselect_objs (siehe Seite 69) in der except-Klausel alle Objekte der Szene deselektiert. Danach wird mit dem Operator bpy.data.groups.new die fehlende Gruppe erstellt.

```
def init_groups():

    try:
        if bpy.data.groups[group_name]:
            pass
    except:
        deselect_objs(bpy.context)
        bpy.data.groups.new(name = group_name)
```

Tab. 5: Materialien für das Addon, dessen Einstellungsparameter und Zuordnung zu den Objekten. Die Parameter Diffuse **und** Specular **sind als** RGB**-Werte angegeben.**

Name	Diffuse	Specular	Alpha	Farbe	Objektzuordnung
GEO.north	(0.0, 0.0, 1.0)	(1.0, 1.0, 1.0)	0.5		Nordrichtung
GEO.south	(0.0, 1.0, 1.0)	(1.0, 1.0, 1.0)	0.5		Südrichtung
REF.black	(0.0, 0.0, 0.0)	(1.0, 1.0, 1.0)	0.5		Referenzpunkt
REF.orange	(1.0, 0.2, 0.0)	(1.0, 1.0, 1.0)	0.5		Referenzpunkt - erste Gruppe
REF.yellow	(1.0, 1.0, 0.0)	(1.0, 1.0, 1.0)	0.5		Referenzpunkt - zweite Gruppe
TOPO.node	(0.3, 0.5, 0.3)	(1.0, 1.0, 1.0)	0.5		Internodium
TOPO.leaf	(0.0, 0.168, 0.0)	(1.0, 1.0, 1.0)	0.5		Blatt
TOPO.flower	(1.0, 0.0, 1.0)	(1.0, 1.0, 1.0)	0.5		Blüte
TOPO.fruit	(1.0, 0.0, 0.0)	(1.0, 1.0, 1.0)	0.5		Frucht

In der *init_groups*-Funktion werden alle Gruppen erstellt, die nicht in dem Scan der Topologie der Pflanze, bzw. bei der Beschreibung des Modells nach dem MTG-Format gebraucht werden. Zu den hier definierten Gruppen gehören:

- Calibration
- Geo_direction
- Ground_slope
- Ref_gr_1
- Ref_gr_2
- Volume

Die Funktionen *set_initial_mtg_lists* und *set_mtg_enum* prüfen und passen die globalen Listen an, die für die Verwaltung der MTG-Strukturen in dem Panel *FASTRAK: Advanced MTG Editor* bearbeitet werden können. Die erste Funktion sucht nach schon vorhandenen MTG-Gruppen für eine Verzeigung, eine Wachstumseinheit oder eine Pflanze. Die Namen der gefundenen Gruppen werden jeweils in einer der drei Listen *branch_list*, *unit_list*, *plant_list* abgespeichert. Die Listen werden für das Definieren der Enum-Eigenschaften *fk_b_list*, *fk_u_list* oder

fk_p_list bei dem Aufruf der set_mtg_enum-Funktion gebraucht. Die MTG-Enum-Eigenschaften werden als Comboboxen in der Benutzeroberfläche dargestellt. Deshalb wird in der Funktion set_mtg_enum, wenn die entsprechende Liste länger als Null ist, der letzte Eintrag als aktiv gekennzeichnet.

```
def set_mtg_enum(case, scn = None):
    global branch_list, unit_list, plant_list

    if case == "branch":
        bpy.types.Scene.fk_b_list = EnumProperty(items = branch_list)

        if len(branch_list) > 0 and scn:
            scn.fk_b_list = str(scn.fk_mtg.b_count - 1)

    # similarly for other unit and plant groups...
```

Erstellung und Verwaltung der Materialeigenschaften

Die nächste Gruppe der Funktionen in dem f_controller-Modul hilft bei der Benutzung der Materialeigenschaften in Blender. Die Funktionen werden vor allem bei der Erstellung der Objekte gebraucht. Sie sind aber auch bei der Initialisierung des Addons wichtig (siehe Seite 60). Zu den Funktionen gehören:

- make_material
- set_material
- find_mat
- find_ob_mat_idx

Der Aufruf der make_material-Funktion braucht vier Argumente, um mit den Befehlen von der Blender-API ein Material zu erstellen. Die Funktion liefert nichts zurück, da das erstellte Material in den Blender-Strukturen abgespeichert wird.

```
def make_material(name, diffuse, specular, alpha):

    mat = bpy.data.materials.new(name)

    mat.diffuse_color = diffuse

    mat.specular_color = specular

    mat.alpha = alpha

    # other material parameters...
```

Die nächste Funktion, set_material, ordnet einem übergebenen Objekt das übergebene Material hinzu. Sie wird bei der Erstellung der Objekte benutzt. Die Materialien werden dem Datenblock des Objektes zugeordnet.

```
def set_material(ob, mat):

    ob.data.materials.append(mat)
```

Die Funktion find_mat gibt True zurück, falls der Name des Materials in der Blender-Kollektion bpy.data.materials gefunden wurde. Die Funktion wird bei der Initialisierung des Addons benutzt (siehe Seite 60). Die Funktion find_ob_mat_idx hilft beim Einfärben der Polygone der Referenzpunkte. Sie liefert aus den Daten des Objektes und aus dem Namen des Materials, die Position des Materials unter der das Material in den Objektdaten abgespeichert wurde. Die Information wird gebraucht, da die Referenzpunkte zur besseren Übersicht mit zwei Farben markiert werden. Um dies zu erreichen, muss man die Position des Materials auf der Materialienliste des jeweiligen Objekts (obj.data.materials) kennen. Der in der Liste gefundene Index des Materials wird im f_obj_factory.py-Modul in der Funktion build_ref den entsprechenden Polygonen des Objekts zugewiesen.

```
for idx in polygon_list:
    obj.data.faces[idx].material_index = mat_idx
```

Update-Funktionen der Blender-Eigenschaften des Addons

Die Update-Funktionen werden in Blender zur Aktualisierung der Eigenschaften benutzt. Sie haben deshalb alle zwei feste Aufrufargumente und immer den Rückgabewert None (siehe Seite 30). Die Zuordnung der Funktionen zu den Eigenschaften, zusammen mit einer kurzen Beschreibung, ist in der Tab. 6 aufgeführt. Detaillierter werden in diesem Abschnitt nur die Funktionen für das Aktualisieren des Durchmessers der Topologieobjekte behandelt.

Die Änderungen des Durchmessers sind für zwei beim Scannen der Topologie entstehende Formen möglich. Für Kugeln oder Zylinder kann im FASTRAK: Object Information-Panel der Durchmesser nachträglich geändert werden. Die Objekte in Blender haben standardmäßig eine Eigenschaft scale. Bei Angabe des Durchmessers für die Kugel wird die Funktion update_sph_diameter aufgerufen. In der Funktion wird zuerst getestet, ob das Objekt eine Kugel ist und falls das zutrifft, wird der Parameter scale des Objektes in jede Richtung auf den ausgewählten Radius gesetzt. Dies ist möglich, da beim Erstellen einer Kugel, die scale- und size-Faktoren auf 1 gesetzt sind und erst dann die Größe des Objekts angepasst wird.

```
# only for sphere
radius = obj.fk_o.diameter/2
obj.scale = [radius, radius, radius]
```

Die Zylinder werden sofort mit dem vom Benutzer gewählten Durchmesser erstellt. Deshalb wird hier zur Skalierung die resize-Funktion benutzt. Sie kapselt den Aufruf des Blender-Operators bpy.ops.transform.resize (Seite 69) ab. Der Operator braucht einen Skalierungsfaktor alpha, um das aktive Objekt zu verändern. Der Durchmesser des Zylinders kann in dem Addon auf zwei verschiedene Arten bearbeitet werden. Die Funktion update_cyl_d0 ändert den Durchmesser gleich für den ganzen Zylinder. Die Funktionen update_cyl_d1 und update_cyl_d2

skalieren den Durchmesser für den oberen, bzw. unteren Deckel des Zylinders. In jeder der drei Funktionen wird der Skalierungsfaktor alpha aus dem vom Benutzer angegebenen Durchmesser und dem Abstand der Vertices, die den aktuellen Radius bestimmen, berechnet (Abb. 17).

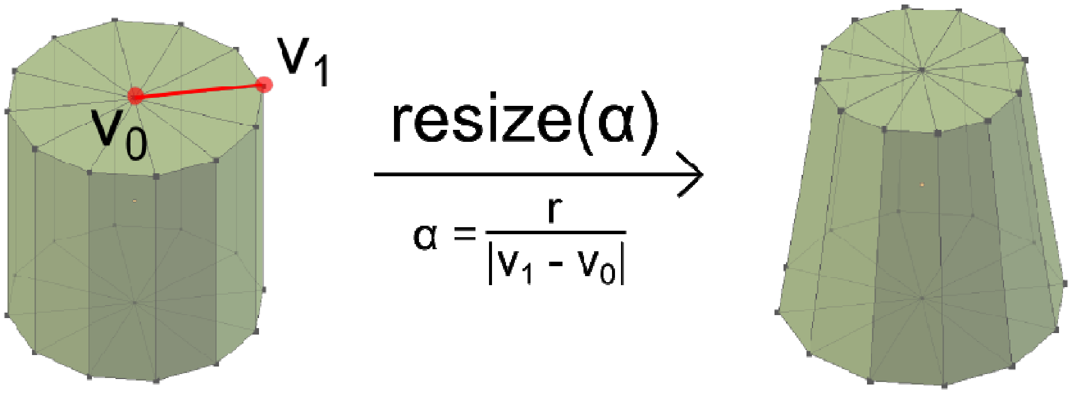


Abb. 17: Skalierung des oberen Deckels des Zylinders. Der Deckel kann nach der Berechnung des Skalierungsfaktor α und Markierung der Vertices des oberen Deckels mit der Funktion resize **geändert werden. Der neue Radius (**r**) wird aus der Benutzereingabe des Durchmessers ermittelt.**

Tab. 6: Update-Funktionen des f_controller.py**-Moduls mit den Namen der Eigenschaften, bei der die jeweilige Funktion aufgerufen wird, und die Beschreibung der Funktionalität.**

Funktion	Eigenschaft	Kurze Beschreibung
update_started	fk_contr.started	Startet den bpy.ops.fk.scan_manager_operator und gibt ein entsprechendes Tonsignal aus.
update_scan_type	fk_contr.scan_type	Ruft den Operator für den Wechsel zwischen dem kontinuierlichen und dem Einzelpunktmodus des Scanners auf.
update_adj_type	fk_adj.type	Ruft den Operator für den Wechsel zwischen dem kontinuierlichen und dem Einzelpunktmodus des Scanners auf. Einzelpunktmodus erforderlich.
update_al_type	fk_al.type	Ruft den Operator für den Wechsel zwischen dem kontinuierlichen und dem Einzelpunktmodus des Scanners auf. Einzelpunktmodus für geographische Richtung und kontinuierlicher Modus für die Neigung erforderlich.
update_topo_type	fk_topo.type	Ruft den Operator für den Wechsel zwischen dem kontinuierlichen und dem Einzelpunktmodus des Scanners auf. Einzelpunktmodus für Standardtopologie und kontinuierlicher Modus für die zusätzliche Topologie erforderlich.
update_contr_increment	fk_contr.increment	Reagiert auf die Änderung des Wertes für das Sendeabstandsinkrement in der Oberfläche. Ruft den bpy.ops.fk.init_update_increment_operator auf.

update_sph_diameter	fk_o.diameter	Setzt einen anderen Durchmesser für die Kugel (Objekt aus dem Topologiescan).
update_cyl_d0	fk_o.cyl_d0	Setzt einen anderen Durchmesser für den ganzen Zylinder (Objekt aus dem Topologiescan).
update_cyl_d1	fk_o.cyl_d1	Setzt einen anderen Durchmesser für den unteren Deckel des Zylinders (Objekt aus dem Topologiescan).
update_cyl_d2	fk_o.cyl_d2	Setzt einen anderen Durchmesser für den oberen Deckel des Zylinders (Objekt aus dem Topologiescan).
update_density	fk_o.density	Berechnet Masse des Objektes in Abhängigkeit von der Dichte. Einsetzbar nur bei den Objekten, die durch Abmessung des Volumens entstanden sind.
update_basis_node	fk_mtg.basis_node	Aktualisiert die Anzeige der Information über der zuletzt gesetzten Verzweigung.
update_succ_axis	fk_mtg.succ_axis	Verursacht das Verstecken der Information über die Verzweigung in dem MTG-Panel.
update_fk_u_members	fk_mtg.activ_u_members	Aktualisiert die Information über die Anzahl der Mitglieder der Gruppe, die auf der Unit-Liste in der Benutzeroberfläche ausgewählt ist.
update_fk_p_members	fk_mtg.activ_p_members	Aktualisiert die Information über die Anzahl der Mitglieder der Gruppe, die auf der Plant-Liste in der Benutzeroberfläche ausgewählt ist.

Hilfsfunktionen für den Topologiescan

Alle abgetasteten Objekte, die in dem Topologiescan entstehen, werden nach dem MTG-Format benannt (siehe Seite 128). Die Benennung ist dabei auf die Internodien-Skala begrenzt. Weitere Abstufungen des Modells kann man mit den FASTRAK: Advanced MTG editor-Panel ergänzen (Abb. 62, Abb. 63 und Abb. 64). Die Konvention für die Erstellung der Namen ist in Tab. 2 dargestellt. Dank der Tatsache, dass die Namen nur an einer Stelle des Addons erstellt und verwaltet werden, ist eine eventuelle Fehlerbehebung oder Änderung der Benennung einfacher durchzuführen.

Die Übersicht der Funktionen, die bei der Benennung und Verwaltung der Topologieobjekte helfen, ist in Tab. 8 enthalten. Die einzige Funktion in der Gruppe, die nicht nur der Namensverwaltung oder der Informationsgewinnung aus dem Namen des Objektes dient, ist die Funktion set_node_type_and_mat. Sie ordnet ein Material einem Objekt zu und setzt den Typ-Flag des Objektes. Der Flag wird in der fk_o.type-Eigenschaft abgespeichert (Tab. 16) und wird in der Benutzeroberfläche im Panel FASTRAK: Object information zum korrekten Zeichnen der Komponenten benutzt.

Tab. 7: Benennung der Internodien in dem Scan der Topologie. A **steht für Axis. Die Buchstabe** N **steht für Internodium und wurde zwecks besserer Lesbarkeit in Blender statt des Buchstabens** I**, der normalerweise im MTG-Format eingesetzt wird, gewählt.** LF **steht für** leaf**.** FL **steht für** flower**.** FR **steht für** fruit**.**

Typ	Benennung
Internodium	A[Ordnungsnummer]/N[Ordnungsnummer]
Blatt	A[Ordnungsnummer]/N[Ordnungsnummer]/LF
Blüte	A[Ordnungsnummer]/N[Ordnungsnummer]/FL
Frucht	A[Ordnungsnummer]/N[Ordnungsnummer]/FR

Tab. 8: Hilfsfunktionen für die Verwaltung der Benennung der Topologieobjekte.

Funktion	Argumente	Rückgabewert	Beschreibung
cr_axis_name	- Blender-Scene (bpy.context.scene)	- Name der neuen Achse	Erstellt den Namen für die neue Achse
cr_node_name	- Blender-Scene	- Name für das neue Objekt	Erstellt den Namen für das neue Objekt
node_name_to_axis_name	- Name des Objekts	- Name der Achse	Gewinnt den Namen der Achse aus dem Namen des Objektes
node_name_to_node_nr	- Name des Objekts	- Ordnungsnummer des Objekts	Gewinnt aus dem Namen des Objekts dessen Ordnungsnummer
node_name_to_axis_nr	- Name der Achse	- Ordnungsnummer der Achse	Gewinnt aus dem Namen des Objekts die Ordnungsnummer der Achse

axis_name_to_axis_nr	- Name der Achse	- Ordnungsnummer der Achse	Gewinnt aus dem Namen der Achse die Ordnungsnummer der Achse
get_infos_from_node_name	- Name des Objekts	- Ordnungsnummer der Achse - Ordnungsnummer des Objekts - Name der Achse - Typ des Objekts	Gewinnt aus dem Namen des Objekts die Ordnungsnummer der Achse, Ordnungsnummer des Objekts, Name der Achse und Typ des Objekts ('LF', 'FL', 'FR')
get_nr_name_type_from_node	- Name des Objekts	- Ordnungsnummer des Objekts - Name des Objekts ohne Achsenamen - Typ des Objekts	Gewinnt aus dem Namen des Objekts die Ordnungsnummer des Objekts, Name des Objekts ohne Namen der Achse und Typ des Objekts ('LF', 'FL', 'FR')
set_node_type_and_mat	- Object - Blender-Scene	kein	Setzt das Flag für den Typ des Objektes und das entsprechende Material (Tab. 5)

Hilfsfunktionen für das Editieren der MTG-Struktur

Die Listen der angelegten Gruppen für die Verzweigungen, Wachstumseinheiten und Pflanzen werden in den Comboboxen des Panels FASTRAK: Advanced MTG editor angezeigt. Die Comboboxen erstellt man in Blender mit Hilfe der Struktur EnumProperty, die eher als statische Struktur gedacht ist. Wenn der Benutzer neue Gruppen anlegt, muss diese dynamisch ständig angepasst werden. Deshalb wurden die in Tab. 9 aufgelisteten Funktionen definiert. Für jede der Funktionen, die einen Namen der Gruppe zu der entsprechenden Enum-Eingenschaft hinzufügen oder entfernen, wird auf der zusätzlichen globalen Python-Liste, die in dem Modul f_controller.py definiert ist, genauso ein Name der Gruppe hinzugefügt und entfernt. Danach wird die entsprechende Enum-Eigenschaft mit der neuen Enum-Liste überschrieben. Die für das Definieren verantwortliche Funktion heißt set_mtg_enum. Beim Überschreiben wird auch der dem Benutzer sichtbare Eintrag der Combobox neu gesetzt.

```
global branch_list

# new combo box for the branches
bpy.types.Scene.fk_b_list = EnumProperty(items = branch_list)

# show the last branch on the list in the combo box in user interface
scn.fk_b_list = str(scn.fk_mtg.b_count - 1)
```

Tab. 9: Hilfsfunktionen für den FASTRAK: Advanced MTG editor**; Argumente, Rückgabewerte und Beschreibung.**

Funktion	Argument(e)	Rückgabewert(e)	Beschreibung
mtg_append_item	- Typ der Aufzählung - Blender-Scene (bpy.context.scene)	---	Fügt eine neue Gruppe am Ende der Liste des angegebenen Typs ('branch', 'unit', 'plant') ein und aktualisiert die Combobox.

mtg_pop_item	- Typ der Aufzählung - Position des Eintrags in der Aufzählung - Blender-Scene	---	Entfernt einen Eintrag aus der angegebenen Position von der globalen Liste des angegebenen Typs und aktualisiert die Combobox.
set_mtg_enum	- Typ der Aufzählung - Blender-Scene	---	Aktualisiert die Combobox mit den Einträgen aus der globalen Liste des gewählten Typs.
mtg_sort_enum	- Typ der Aufzählung - Blender-Scene	---	Sortiert alphabetisch die Liste des gewählten Typs und aktualisiert die Combobox.
mtg_names_from_branch	- Name der Verzweigung	- Name des Basisobjekts in der Verzweigung - Name der nachfolgenden Achse	Extrahiert den Namen des Basisobjektes und der Nachfolgerachse aus dem Namen der Verzweigung.
mtg_key_to_item_name	- Position des Eintrags in der Aufzählung - Typ der Aufzählung	- Name der MTG-Gruppe	Konvertiert die Position auf der Liste des gewählten Typs in den Namen der MTG-Gruppe.
mtg_name_to_key	- Name der MTG-Gruppe - Typ der Aufzählung	- Position des Eintrags in der Aufzählung	Konvertiert den Namen der MTG-Gruppe auf der Liste des gewählten Typs in die Position des Eintrags.
get_first_name_in_axis	- Name der Achse	- Namen des ersten Objekts auf der Achse	Findet den Namen des ersten Objekts auf einer Achse (A[Ordnungsnr.]).

Kapselung oft benutzter Blender-Operatoren

Die letzte Gruppe der Hilfsfunktionen, die im f_controller.py definiert ist, kapselt die Aufrufe der Blender-Operatoren, die sehr häufig beim Erstellen und Manipulieren der Scanobjekte benutzt werden, ab. Die Art und Weise der Implementierung für die Gruppe der Funktionen ist sehr ähnlich, deshalb wird nur eine Funktion näher erklärt. Alle anderen Funktionen sind in Tab. 10 aufgelistet und beschrieben. Alle Blender-Operatoren, die bei den Funktionen benutzt wurden, sind in der Dokumentation zur Blender-API beschrieben (Blender Foundation 2012).

Die Funktion resize wird mit einem Skalierungsfaktor alpha aufgerufen. Dieser Faktor entscheidet, um wie viel ein Objekt vergrößert oder verkleinert wird. Bei Aufruf der Funktion wird zuerst mit Hilfe der Funktion poll getestet, ob der Operator für Änderung des Editierungsmodus benutzt werden kann. Falls ja, wird in den Edit-Modus gewechselt. Danach werden die Werkzeuge für das Selektieren der Teile eines Gitters so eingestellt, dass nur die Vertices des Gitters selektiert werden können. Nach den Vorbereitungen kann der Operator für das Skalieren der selektierten Vertices ausgeführt werden. Das selektierte Gitter wird in x-, y- und z-Richtung um den gleichen Faktor skaliert. Am Ende setzt die Funktion resize den Editierungsmodus in den Objektmodus zurück.

```
def resize(alpha):
    # validate the context
    if bpy.ops.object.mode_set.poll():

        # set 3D-View properties
        bpy.ops.object.mode_set(mode = 'EDIT')
        bpy.context.tool_settings.mesh_select_mode = (True, False, False)

        # resize active object
        bpy.ops.transform.resize(value = (alpha, alpha, alpha))

        # go back to standard mode for the scan
        bpy.ops.object.mode_set(mode = 'OBJECT')
```

Tab. 10: Funktionen für die Kapselung der Blender-Operatoren.

Funktion	Argument(e)	Beschreibung
deselect_mesh	---	Deselektiert das Gitter des Objekts.
resize	Skalierungsfaktor	Skaliert das Gitter des Objektes.
delete_vert	---	Löscht alle selektierte Vertices.
add_face	---	Erstellt ein Polygon, falls möglich, aus den selektierten Vertices.
select_mesh	Wahrheitswert für Wechsel zwischen Objekt- und Editiermodus.	Selektiert das ganze Gitter. Je nach dem Wahrheitswert des Arguments bleibt der 3d-View in dem Editiermodus (False) oder wechselt wieder in den Objektmodus (True).
unify_normals	---	Berechnet die Normalenvektoren neu, so dass alle in eine Richtung zeigen.
select_all_from_group	Name der Gruppe	Selektiert alle Objekte der ausgewählten Gruppe.
deselect_objs	bpy.context	Deselektiert alle Objekte in der Scene
select_objs	bpy.context	Selektiert alle Objekte in der Scene

Das Modul f_labels.py

In dem Modul sind alle Beschriftungen, die auf der Benutzeroberfläche sichtbar sein könnten, zusammen gestellt. Kontextabhängig werden nicht benötigte Teile aus dem Benutzerinterface ausgeblendet. Das Abtrennen der Strings für den Benutzer vom Quelltext ist nicht zwingend erforderlich. Sie beschleunigt aber eine Korrekturen und Änderungen, bzw. Übersetzung der Beschriftungen in der Oberfläche.

Damit in anderen Modulen die hier definierten Strings eingesetzt werden können, wird am Anfang jedes Moduls die Klasse StringFactory importiert. Danach wird das globale Objekt lbs erstellt. Die Klasse StringFactory besitzt auch eine Hilfsfunktion

für die Abbildung einer ganzen Zahl, die den Objekttyp definiert, in den Namen des Objekttyps (Tab. 11).

Tab. 11: Abbildung des ganzzahligen Typs des Objekts auf dessen Beschriftung in der Benutzeroberfläche.

Ganzzahliger Typ des Scanobjekts	UI-Bezeichnung
1	Calibration geometry
2	Reference point
3	Geographical direction
4	Slope to XY plane
5	Internode
6	Internode - Leaf
7	Internode - Flower
8	Internode - Fruit
9	Volume
10	Blender object without scan data!

Das Modul f_obj_factory.py

In diesem Modul befinden sich alle Funktionen, die zum Zeichnen der gescannten Gegenstände nötig sind. Auf die wichtigsten theoretischen und technischen Grundlagen der Implementierung des Zeichnens der gescannten Objekte wird nach Bedarf einzeln eingegangen. Die Besonderheiten bei der Implementierung des Zeichnens der gescannten Objekte werden am Ende des Kapitels angesprochen. Alle Funktionen des f_obj_factory.py-Moduls sind in Tab. 12 mit einer kurzen Beschreibung zusammengestellt.

Tab. 12: Die Funktionen für das Zeichnen der gescannten Objekte, deren Argumente, Rückgabewerte und Beschreibung der Funktionalität.

Funktion	Argument(e)	Rückgabewert(e)	Beschreibung
build_calibr_box	- Blender-Kontext (bpy.context) - Blender-Scene (bpy.context.scene) - Liste mit den aufgenommenen Koordinaten und Rotationen - Zeit des Ankommens der vorherigen Datenzeile	Flag für das Zeichnen des Gitters (fk_contr.draw_mesh), immer False	Erstellt das Gitter für die Kalibrierungsgeometrie. Setzt deren Eigenschaften. *(Einzelpunktmodus)*

build_ref	- Blender-Kontext - Blender-Scene - Liste mit den aufgenommenen Koordinaten und Rotationen - Zeit des Ankommens der vorherigen Datenzeile	Flag für das Zeichnen des Gitters (fk_contr.draw_mesh), immer False	Erstellt die Objekte, die die Referenzpunkte darstellen. Setzt deren Eigenschaften. *(Einzelpunktmodus)*
build_geo	- Blender-Kontext - Blender-Scene - Liste mit den aufgenommenen Koordinaten und Rotationen - Zeit des Ankommens der vorherigen Datenzeile	Flag für das Zeichnen des Gitters (fk_contr.draw_mesh), immer False	Erstellt die Objekte, die Norden und Süden in dem aktuellen Koordinatensystem definieren. Setzt deren Eigenschaften. *(Einzelpunktmodus)*
build_slope	- Blender-Kontext - Blender-Scene - Liste mit den aufgenommenen Koordinaten und Rotationen	Flag für das Zeichnen des Gitters (fk_contr.draw_mesh)	Zeichnet das Objekt, das die Neigung zu der XY-Ebene darstellt. Setzt deren Eigenschaften. *(kontinuierlich)*
rotation_matrix	- Rotationsachse - Rotationswinkel	Rotationsmatrix	Erstellt die Rotationsmatrix für das Zeichnen eines Zylinders.
build_std_topo	- Blender-Kontext - Blender-Scene - Liste mit den aufgenommenen Koordinaten und Rotationen - Zeit des Ankommens der vorherigen Datenzeile	Flag für das Zeichnen des Gitters (fk_contr.draw_mesh), immer False	Erstellt, je nach Einstellung in der Benutzeroberfläche, die Geometrie für die Vermessung der standardmäßigen Topologie (Linie, Zylinder oder Kugel). Setzt deren Eigenschaften. *(Einzelpunktmodus)*
build_free_leaf	- Blender-Kontext - Blender-Scene - Liste mit den aufgenommenen Koordinaten und Rotationen	Flag für das Zeichnen des Gitters	Erstellt aus der Punktwolke ein Blatt. Die Punktwolke wird als Fläche dargestellt. Setzt Eigenschaften des Objekts.
build_free_flower	- Blender-Kontext - Blender-Scene - Liste mit den aufgenommenen Koordinaten und Rotationen	Flag für das Zeichnen des Gitters	Erstellt aus der Punktwolke eine Blüte. Die Punktwolke wird als 2D-Objekt betrachtet. Setzt Eigenschaften des Objekts. *(kontinuierlich)*
build_free_fruit	- Blender-Kontext - Blender-Scene - Liste mit den aufgenommenen Koordinaten und Rotationen	Flag für das Zeichnen des Gitters	Erstellt aus der Punktwolke eine Frucht. Die Punktwolke wird als 3D-Objekt dargestellt. Setzt Eigenschaften des Objekts. *(kontinuierlich)*
build_volume	- Blender-Kontext - Blender-Scene - Liste mit den aufgenommenen Koordinaten und Rotationen	Flag für das Zeichnen des Gitters	Erstellt aus der Punktwolke ein 3D-Objekt, bei dem das Volumen und die Masse berechnet werden kann. Setzt Eigenschaften des Objekts. *(kontinuierlich)*

Das für den Scan eingestellte Datenformat (Tab. 3) der ankommenden Datenzeile erlaubt es zu erkennen, ob die Stifttaste gedrückt ist oder nicht. Für das Zeichen im kontinuierlichen Modus wurde in der Anwendung entschieden, dass die Daten nur für eine unbetätigte Stifttaste gezeichnet werden dürfen. Der Grund dafür ist, dass man beim Drücken der Taste die Empfängerspitze kaum ruhig halten kann. Für diese Situation ist es aber nötig, ein Signal für den Anfang des Aufnehmens der Punkte und eines für das Stoppen der Aufzeichnung zu haben. Das Signal für das Starten und Stoppen des Zeichnens wird in der Erweiterung mit zwei Eigenschaften gesteuert. Die erste Eigenschaft heißt fk_contr.fButton_state. Sie speichert die Zustände der Taste für die zwei letzten Punkte. Die zweite Eigenschaft, die bei der Steuerung des kontinuierlichen Zeichnens eine Rolle spielt, heißt fk_contr.draw_mesh. Die beiden Eigenschaften zusammen erlauben es zu erkennen, wann der erste Punkt des Objekts angelegt werden soll, wann die nächsten Punkte des Gitters gezeichnet werden sollen und wann das Zeichnen abgeschlossen werden soll. Wenn der Benutzer die Vermessung über die Stifttaste steuert, wird die Aktualisierung der Zustandsliste mit Hilfe der Funktion __update_fButtonState der Klasse SCENE_OT_fk_scan_manager für jeden gültigen Datensatz durchgeführt.

```
if line_state == r"1\r\n'":
    scn.fk_contr.fButton_state = [scn.fk_contr.fButton_state[1], True]
else:
    scn.fk_contr.fButton_state = [scn.fk_contr.fButton_state[1], False]
```

Dies geschieht, bevor eine Funktion für das Zeichnen eines Objektes aufgerufen wird. Der Benutzer kann aber auch die Vermessung über die Oberfläche steuern. In dem Fall erfolgt die Aktualisierung der Zustandsliste erst am Anfang der jeweiligen Funktion für das Zeichnen eines Objekts. Die Zusammenstellung der Zustände und der Aktionen ist in Tab. 13 dargestellt.

Tab. 13: Zusammenstellung der Zustände der Eigenschaft fk_contr.fButton_state **und** fk_contr.draw_mesh **und der zugehörigen Aktionen bei dem kontinuierlichen Modus des Scans.** A **steht für der Zustand der Taste in der Zeit** $t-1$**.** B **steht für den Zustand der Taste in der Zeit** t **uns** C **für den Zustand der Variable** fk_contr.draw_mesh**.**

A	B	C	Aktion
0	0	0	Die ankommenden Punkte werden nicht gezeichnet.
1	1	1/0	
1	0	0	
0	1	0	Signal für den Anfang des Zeichnens eines Objekts.
1	0	1	Der erste/nächste Vertex des Objekts wird gezeichnet.
0	0	1	
0	1	1	Das Zeichnen wird abgeschlossen.

Das Steuern des Zeichnens im Einzelpunktmodus des Scanners

Der Einzelpunktmodus erfordert im Gegensatz zum kontinuierlichen Modus keine Zustandsvariablen zur Steuerung der Vermessung. Hier ist es aber wichtig, dass, wenn der Benutzer einmal auf die Taste gedrückt hat, egal wie lange der Druck ausgeübt wurde, nur ein Punkt aufgenommen und dargestellt wird. Wie lange verschiedene Personen die Taste gedrückt halten, lässt sich nicht steuern. Man kann aber dieses Ereignis in der Software entsprechend behandeln. Im Addon gibt es dafür in der Klasse SCENE_OT_fk_scan_manager die Variable __point_t_old. Sie wird nach jedem Aufruf einer Zeichenfunktion aktualisiert. Vor dem Aktualisieren wird die Variable der Funktion übergeben, die das Zeichnen des Objekts im Einzelpunktmodus abwickelt. Am Anfang der Funktion wird immer die Differenz zwischen der Zeit des vorherigen Punkts und der aktuellen Zeit geprüft. Wenn die Differenz den Schwellenwert nicht überschreitet, wird der Punkt ignoriert und die Ausführung der Zeichenfunktion endet. Andernfalls kann der Punkt gezeichnet werden.

```
# difference between time t and t-1
if time() - t_old  > .5:
    # draw the object ...
```

Erstellen der Scanobjekte in Blender

In der Erweiterung werden zwei Methoden zur Erstellung von Objekten in Blender ausgenutzt. Die erste ist die Benutzung der vorgefertigten Blender-Routinen, wie z.B. bpy.ops.mesh.primitive_uv_sphere_add, bpy.ops.mesh.primitive_cube_add, bpy.ops.object.text_add usw. Die andere Möglichkeit ist das Zeichen der Geometrie direkt aus den Daten. Hierfür steht die Funktion from_pydata zur Verfügung. Bei der ersten Variante muss man die verschiedenen Argumente der Funktionen, z. B. zum Setzen der Position oder des Durchmessers des Objekts, den Wünschen nach einstellen. Die meisten Parameter haben hier Standardwerte, die auch ohne weiteres übernommen werden können. Die Funktion from_pydata erfordert dagegen, dass alle benötigten Eigenschaften des Objekts vom Programmierer gesetzt werden. Da die Funktion from_pydata in fast allen Zeichenfunktionen des Addons eigesetzt wurde, wird die Art der Erstellung eines Blender-Objekts an dieser Stelle genauer erklärt.

Bevor die Funktion from_pydata aufgerufen werden kann, muss man ein neues Gitterobjekt erstellen. Erst danach kann man die Funktion auf dem neu erstellten Objekt aufrufen. Die Funktion nimmt drei Argumente entgegen. Als erstes muss man eine Liste der Vertices angeben. Das zweite Argument ist für die Liste der Kanten reserviert und das dritte für die Liste der Polygone. Dabei muss man beachten, dass für die Vermeidung von Fehlern bei der Erstellung von Objekten eine der letzten beiden Listen leer bleiben muss. In der Routine werden die fehlenden Daten automatisch ergänzt.

```
# new mesh object
mesh_object = bpy.data.meshes.new(mesh_object_name)

# arguments for the function: vertices_list, edges_list, faces_list
object_mesh.from_pydata(vertices, [], [])

# update the mesh object
object_mesh.update()
```

Als nächstes muss die sichtbare Geometrie erstellt werden. Dafür übergibt man im Aufruf der Funktion bpy.data.objects.new den Namen für das neue Objekt und das Gitter, das dem Objekt gehören soll. Der Unterschied zwischen einem Gitterobjekt und einem normalen Objekt ist, dass das gleiche Gitter zu verschiedenen Objekten gehören kann und das eigentliche Objekt die eindeutige Hülle für das Gitter ist. Teilen sich mehrere Objekte ein Gitter, werden, bei Änderung des Gitters in einem Objekt, alle anderen Objekte die gleiche Form annehmen. Die Objekte, die sich ein Gitter teilen, kann man aber unabhängig voneinander in der 3D-Ansicht positionieren. Erst nach dem Linken zu der Szene wird das Objekt für den Benutzer sichtbar sein.

```
# create new object from mesh
object = bpy.data.objects.new(object_name, object_mesh)

# show the object in the current scene
bpy.context.scene.objects.link(object)
```

Die Objekte werden im kontinuierlichen Modus online erstellt, deshalb ist es wichtig, eine Möglichkeit zu haben, das schon existierende Gitter zu erweitern. Hierfür kann man die Funktion add benutzen. Die Funktion braucht bei dem Aufruf die Information über die Anzahl der Vertices, die hinzugefügt werden sollen. Danach muss man an entsprechender Stelle in der Liste der Vertices des Gitters die neuen Koordinaten angeben.

```
# add one vertex to the mesh
object.data.vertices.add(1)

# set the coordinates for this vertex
object.data.vertices[-1].co = [x, y, z]
```

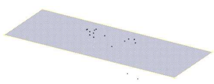
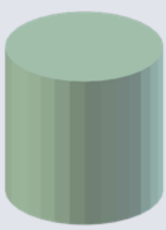
Zum Ende des Zeichnens, wenn der letzte Punkt aufgenommen wurde, wird in jeder Zeichenfunktion des fk_obj_factory.py-Moduls der sogenannte Ursprung (Origin) des Objektes zum Schwerpunkt des Gitters verschoben. Dies darf nicht vor dem Abschluss des Zeichnens passieren. Standardmäßig befindet sich der Ursprung zuerst im Punkt (0, 0, 0). Die Funktion from_pydata zeichnet die angegebenen Koordinaten bezüglich dieses Punktes. Wenn man zu früh den Ursprung verschiebt, verursacht dies eine Verschiebung des gezeichneten Objekts.

```
# shift the origin to the center o the shape
bpy.ops.object.origin_set(type = 'ORIGIN_GEOMETRY')
```

Speicherung der Informationen zu den gescannten Objekten

Die Vermessung erfordert, dass zu den abgetasteten Objekten auch zusätzliche Informationen gespeichert werden. Deshalb werden während und nach der Erstellung eines Scanobjekts für die Objekte entsprechende Scaneigenschaften aktualisiert. Dies geschieht in jeder der Zeichenfunktionen des Moduls. Die Liste aller bei einem Objekt zur Verfügung stehenden Eigenschaften zusammen mit deren Beschreibung befindet sich in weiteren Abschnitten des Kapitels (siehe Seite 99). Die Aufteilung der Objekteigenschaften unter Berücksichtigung der Objekttypen ist in Tab. 14 dargestellt.

Tab. 14: Eigenschaften der Objekte, die in dem Panel FASTRAK: Object information **angezeigt werden.**

Objekt	Beispielbild	Angezeigte Eigenschaften	Editierbar
Kalibrierungsbox		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Verschiebungsvektor, gemessen vom Ursprung des Senders zum Ursprung der Box	-
Referenzpunkt		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Der Messpunkt (x, y, z, yaw, pitch, roll)	-
Geographische Richtung		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Der Messpunkt (x, y, z, yaw, pitch, roll)	-
Neigung gegen XY-Ebene		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Neigung zu der XY-Ebene des Senders	-
Internodium - Zylinder		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Länge in Zentimeter	-

		Durchmesser (für oberen und/oder unteren Deckel)	✓
		Der erste Messpunkt (x, y, z, yaw, pitch, roll)	-
		Der zweite Messpunkt (x, y, z, yaw, pitch, roll)	-
Internodium - Linie		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Länge in Zentimeter	-
		Der erste Messpunkt (x, y, z, yaw, pitch, roll)	-
		Der zweite Messpunkt (x, y, z, yaw, pitch, roll)	-
Internodium - Sphäre		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Länge in Zentimeter	-
		Durchmesser	✓
		Der Messpunkt (x, y, z, yaw, pitch, roll)	-
Internodium – Blatt (Blüte), kontinuierlicher Scan		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Fläche des Blattes	-
Internodium – Frucht, kontinuierlicher Scan		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
Vermessung des Volumens		Kommentar	✓
		Typ des Objekts	-
		Datum und Zeit der Erstellung	-
		Volumen	✓
		Dichte	✓
		Masse	-

Geometrie für die Kalibrierung der Daten

Die Referenzbox für das Kalibrieren wird im Einzelpunktmodus abgemessen. Es müssen dafür 48 Punkte in 3 Ebenen mit jeweils 16 Punkten in festgelegter Reihenfolge aufgenommen werden (siehe Seite 123). Während der Messung werden die Referenzpunkte mit den from_pydata und vertex.add Funktionen in der 3D-Ansicht gezeichnet. Wenn alle Punkte erfasst wurden, wird das Gitter gezeichnet. Anschließend werden die Koeffizienten für die Kalibrierung der ankommenden Koordinaten ermittelt (siehe Seite 95). Wie gut die Kalibrierung geworden ist, kann man in der 3D-Ansicht beim Vergleich der Modellbox und der kalibrierten Box feststellen (Abb. 18).

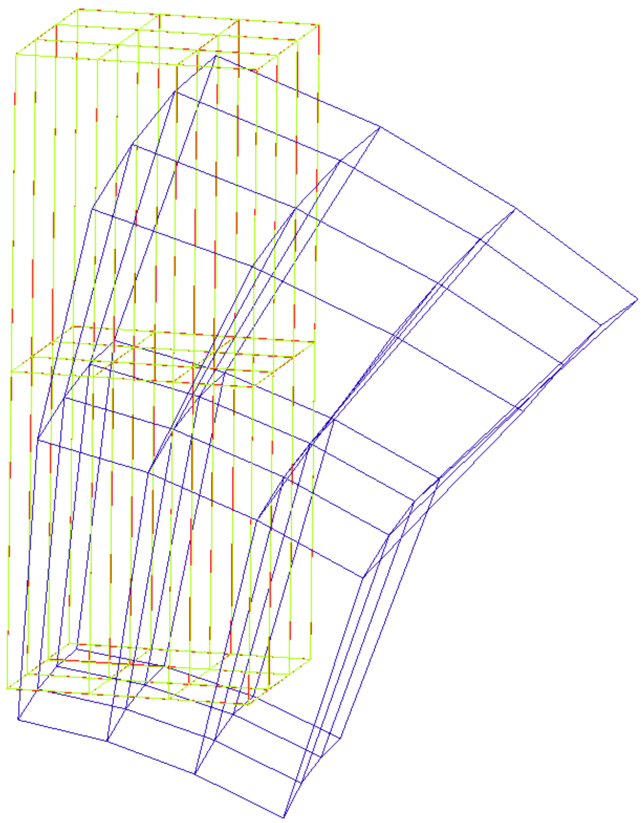


Abb. 18: Kalibrierung der Referenzbox. Die vermessene Box ist blau markiert. Die korrigierte Box ist hellgrün markiert. Die Modellbox ist rot markiert und sichtbar nur an den Stellen, an denen die korrigierte Box von dem Modell abweicht.

Zeichnen der Referenzpunkte

Die Vermessung einer Pflanze muss nicht unbedingt in einer einzigen Sitzung stattfinden. In dem Addon gibt es die Möglichkeit, nach einer Veränderung der Position des Senders die nächsten Punkte an die schon mit einer anderen Position des Senders aufgenommenen Daten anzupassen. Die Voraussetzung dafür ist das Vermessen der vier Referenzpunkte vor der Verschiebung des Senders und der vier Punkte nach der Verschiebung des Senders. In der Funktion build_ref wird kontrolliert, wie viele Punkte aus der ersten und der zweiten Gruppe schon aufgenommen wurden. Damit man die Punkte auf dem Bildschirm sofort unterscheiden kann, sind sie farblich von allen anderen Objekten und von sich selber abgesetzt (Abb. 19).

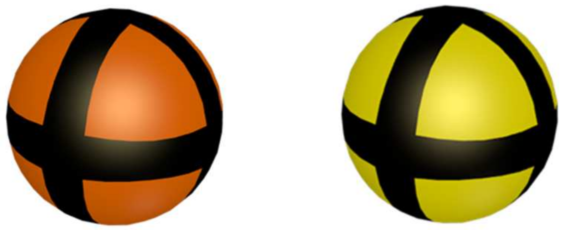


Abb. 19: Einfärbung der Referenzpunkte. Die Punkte der ersten Gruppe (Anfangszustand vor der Verschiebung) sind schwarz-orange eingefärbt. Die Punkte der zweiten Gruppe (Zustand nach der Verschiebung) sind schwarz-gelb.

Die Erstellung solcher Objekte in Blender muss in mehreren Schritten gemacht werden. Der Grund dafür ist, dass der Index der Polygone, die eingefärbt werden sollen, angegeben werden muss. Man kann das z. B. erreichen, wenn man zuerst einen Standardwürfel von Blender mit bpy.ops.mesh.primitive_cube_add einfügt und diesen danach mit Modifizier-Operatoren, wie SUBSERF, shade_smooth, tosphere modifiziert, bis der Würfel in eine Sphäre übergeht (Abb. 20). Danach können zu der Liste der bekannten Polygone die entsprechenden Materialien hinzugefügt werden. Wenn alle vier Punkte gesetzt sind, hat der Benutzer die Möglichkeit, die Entzerrungsmatrix für das Zeichnen der neuen Punkte zu berechnen. Die Berechnung wird mit bpy.ops.fk.adj_calc_equalization_operator durchgeführt. Die Beschreibung der Berechnung ist in dem Absatz über das fk_operators.py-Modul zu finden (siehe Seite 94). Die Ein- und Ausschaltung der Korrektur für die aktuell ankommenden Punkte kann der Benutzer über das Kontrollpanel nach der Berechnung der Entzerrungsmatrix steuern.

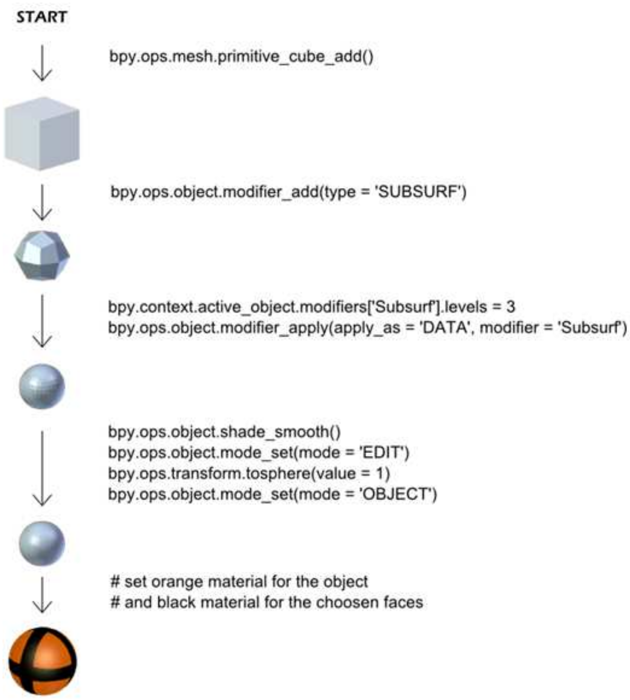


Abb. 20: Schritte zur Erstellung der Geometrie für die Darstellung eines Referenzpunktes. Der Beispielpunkt gehört zur ersten Referenzgruppe, die die Punkte vor der Verschiebung des Senders beinhaltet.

Darstellung der geographischen Nord-Süd-Richtung

Die Punkte für die geographischen Richtungen werden im Einzelpunktmodus des Scanners aufgenommen. Nord und Süd sind mit Hilfe der Blender-Textobjekte im 3D-Fenster dargestellt. Ein Textobjekt wird mit dem Operator bpy.ops.object.text_add zu der Szene hinzugefügt.

```
bpy.ops.object.text_add(location = point, rotation = (-pi/2, 0, 0))
```

Die aufgenommenen Koordinaten bestimmen die Platzierung des Ursprungs des Objekts. Standardmäßig wird ein Textobjekt so ausgerichtet, dass der Text in der Top-Ansicht des 3D-Fensters lesbar ist. Dabei zeigt die positive z-Achse zum Benutzer. Da die positive z-Achse des Senders nach unten zeigt und der Benutzer meistens die Ausrichtung des 3D-Fensters dementsprechend anpasst, wird in dem Addon bei der Erstellung des Textes für die Markierung der geographischen Richtungen die entsprechende Rotation angegeben.

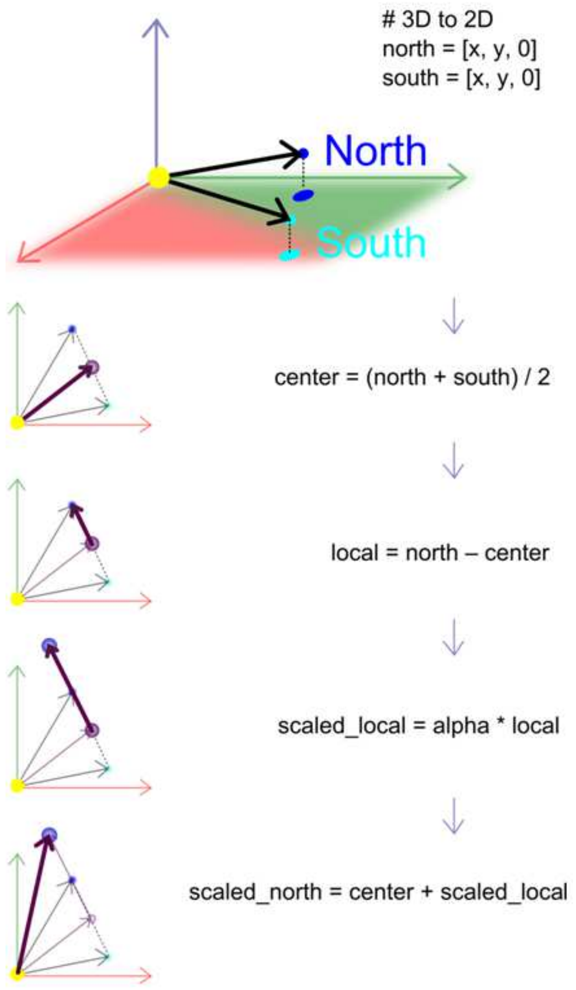


Abb. 21: Zeichnen der Objekte für das Andeuten der Nord-Süd-Ausrichtung des gescannten Raumes. Die x-Achse ist rot, die y-Achse grün und die z-Achse blau markiert.

Mit dem Attribut der Objektdaten body wird die Beschriftung, die in dem 3D-Fenster sichtbar ist, geändert. Danach wird eine Reihe von anderen Attributen des Textobjekts entsprechend gesetzt.

```
# the text on the screen
bpy.context.active_object.data.body = 'New label'
# other text attributes...
```

Wenn die zwei Punkte für die Nordrichtung und für die Südrichtung angegeben und die entsprechenden Objekte erstellt wurden, wird die z-Koordinate bei beiden Punkten abgeschnitten. Danach werden die entsprechenden Punkte skaliert, so dass der Text am Rande des vom Trackingsystem messbaren Raumes in den gewählten Richtungen in der XY-Ebene platziert wird. Zusätzlich zu dem Text werden Linien von den gescannten, auf die XY-Ebene projizierten Punkten zu den neuen Positionen der Textobjekte geführt. Die Methode zur Skalierung der Vektoren der Nord- und Südrichtung ist in Abb. 21 dargestellt. Nachdem die Berechnungen fertig sind und die Textobjekte richtig platziert wurden, wird der Text mit dem Operator bpy.ops.object.convert zu einem Gitterobjekt konvertiert. Am Ende werden die Metainformationen aus dem Scan zu dem Objekt hinzugefügt.

Zeichnen der Geometrie für die Darstellung der Neigung gegen die XY-Ebene

Der Benutzer hat in dem Addon eine Möglichkeit der Berechnung der mittleren Neigung gegen die XY-Ebene des Senders. Die Punkte für das Ermitteln der Neigung werden im kontinuierlichen Modus des Scanners aufgenommen. Der Rumpf der Funktion build_slope entspricht der Beschreibung des kontinuierlichen Zeichnens (siehe Seite 73). Die Berechnung des Winkels wird mit dem Operator bpy.ops.fk.al_calc_slope_operator durchgeführt. Die Beschreibung der Methode folgt auf Seite 96. Das für den Benutzer sichtbare Ergebnis der Punktaufnahme ist eine viereckige Ebene. Sie wird mit der Funktion from_pydata erstellt (siehe Seite 74). Ob die Ebene ausreichend gut an die Punktwolke angepasst wurde, kann man im Editiermodus des 3D-Fensters feststellen (Abb. 22).

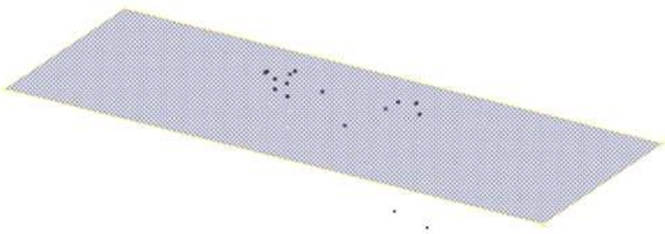


Abb. 22: Eine Ebene für die Darstellung der Neigung gegen die XY-Ebene des Senders und die aufgenommenen Scanpunkte. Sicht im Edit-Modus des 3D-Fensters.

Zeichnen der Standardformen für die Topologie – Zylinder, Linie und Sphäre

Für das Abtasten der Topologie der Pflanzen ist es wichtig, dass eine schematische Darstellung des vermessenen Objekts im Rechner erstellt wird. Die entscheidenden Informationen sind hier die Länge der Internodien und deren Durchmesser. Zusätzlich zu diesen Daten braucht der Benutzer nur noch eine Möglichkeit zur Speicherung der Metainformationen über die Pflanzenteile. Deshalb werden in

diesem Addon als erste Darstellungsmöglichkeit der Struktur der Pflanze einfache Geometriekomponenten wie Sphäre, Linie oder Zylinder zur Verfügung gestellt. Die Anzahl der Scanpunkte, die pro Geometriebaustein benötigt wird, ist somit auf ein Minimum reduziert. Für eine Sphäre wird jeweils nur ein Punkt gebraucht. Für das Zeichnen einer Linie oder eines Zylinders sind entweder zwei oder ein Punkt nötig. Dies verkürzt die Dauer der Aufnahme der Daten.

Zum Zeichnen einer Sphäre wird in der Funktion build_std_topo ein Standardoperator von Blender, bpy.ops.mesh.primitive_uv_sphere_add, benutzt. In dem Aufruf des Operators werden außer der Position des Objektes zwei Parameter, die über die Benutzeroberfläche einstellbar sind, angegeben. Die Sphäre wird somit um den aufgenommenen Punkt gezeichnet.

```
bpy.ops.mesh.primitive_uv_sphere_add(segments = scn.fk_topo.shp_n_seg,
                                     ring_count = scn.fk_topo.shp_n_rings,
                                     location = cData[0:3])
```

Die Sphäre wird zuerst in ihrer Standardgröße gezeichnet. Danach wird mit dem Operator bpy.ops.transform.resize der gewünschte Durchmesser eingestellt.

Die Linie und der Zylinder werden mit der Funktion from_pydata (siehe Seite 74) erstellt. Für das Zeichnen der Linie werden zwei Punkte und eine Kantenverbindung in dem Aufruf angegeben. Die Liste der Polygone bleibt hier leer.

```
# arguments for the function are: list of vertices, list of edges, list of
faces
object_mesh_data.from_pydata([first_point, second_point], [[0, 1]], [])
```

Für das Zeichnen eines Zylinders wird zuerst eine Liste der Vertices und Polygone erstellt, aus denen in der Funktion from_pydata das Gitter erstellt wird. Der Zylinder wird um die zwei aufgenommenen Punkte mit angegebenem Durchmesser gezeichnet. Die Gerade, die man durch die zwei Punkte führt, wird nach der Normierung als Drehachse bei der Erstellung der Rotationsmatrix ausgenutzt. Die Drehachse ist gleichzeitig der Normalenvektor zu den Ebenen, in denen die neuen Punkte des Zylindermantels platziert werden (Abb. 23). Die Rotationsmatrix wird in der Funktion rotation_matrix erstellt. Die Funktion braucht zwei Argumente, die Drehachse n und den Drehwinkel Θ. Die Ausrechnung der Drehmatrix erfolgt nach der Formel (Hübner 2011):

$$R = \begin{pmatrix} a^2+b^2-c^2-d^2 & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & a^2-b^2+c^2-d^2 & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & a^2-b^2-c^2+d^2 \end{pmatrix}.$$

Der Variable a steht für $\cos(\Theta/2)$. Weitere drei Parameter stehen für $b = \sin(\Theta/2)\ n_x$, $c = \sin(\Theta/2)\ n_y$ und $d = \sin(\Theta/2)\ n_z$. Die Summe der Parameter a, b, c und d muss gleich 1 sein. Die Berechnung des ersten Punktes, der zur Erstellung der weiteren Mantelpunkte mit Hilfe der Rotationsmatrix benötigt wird, wird mittels der Gleichung

$$\xi \coloneqq x - o$$

berechnet. Dabei bezeichnet x einen beliebigen Vektor und o den Schwerpunkt einer der Zylinderscheiben. Da die Zylinderscheiben auf einer zu dem Normalenvektor n orthogonaler Ebene liegen, kann dies zur Herleitung der Koordinaten des ersten Punktes auf der Scheibe ausgenutzt werden.

$$(x - o) \cdot n = \xi \cdot n$$

$$n_x \xi_x + n_y \xi_y + n_z \xi_z = 0$$

$$\xi_z = \frac{-1}{n_z}\left(n_x \xi_x + n_y \xi_y\right)$$

Wenn dies bekannt ist, wird ein Vektor R_1 bestimmt. Anhand dessen werden alle anderen Vektoren auf der Scheibe mit Hilfe der Rotationsmatrix ausgerechnet.

$$R_1 = (1, 0,\ \xi_z) = \left(1, 0, -\frac{n_x}{n_z}\right)$$

Am Ende werden alle Vektoren auf die richtige Länge mit dem angegeben Radius R skaliert.

$$R_i = \frac{R}{|R_i|} R_i$$

Der letzte Schritt ist die Umrechnung von den lokalen Koordinaten des Zylinders auf die globalen des Senders.

$$r_i = R_i + o$$

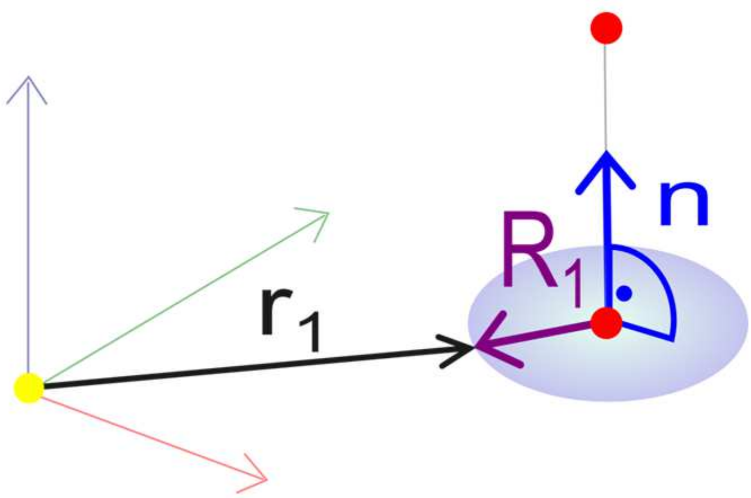


Abb. 23: Erstellung der Punkte auf dem Mantel des Zylinders. Die roten Punkte wurden im Scan aufgenommen.

Zusätzliche Formen – Blatt, Blüte und Frucht

Beim Scannen der Pflanzentopologie kann man, alternativ zu den einfachen geometrischen Formen, die reale Struktur der Blätter, Blüten und Früchte mit Hilfe des Addons in Blender darstellen lassen. In dem Modul fk_obj_factory.py sind dafür drei Funktionen zuständig: build_free_leaf, build_free_flower und build_free_fruit.

Die Funktionen für das Bauen eines Blattes und einer Blüte unterscheiden sich zurzeit nur dadurch, dass am Ende des Erstellens eines Objektes dem Objekt ein anderer Typ zugewiesen wird. Der Grund dafür ist, dass die Vermessung der Blüten in den nächsten Versionen des Programms ausgebaut werden soll. Da die Methoden für das Zeichnen der Objekte bei den beiden Funktionen gleich sind, wird nur die Funktion für die Erstellung der Blätter besprochen. Die Blätter werden in zwei Phasen abgetastet. Zuerst muss der Benutzer den Rand des Blattes abmessen. Danach muss mindestens ein Punkt vom Inneren des Blattes aufgenommen werden. Die Messung wird im kontinuierlichen Modus des Scanners durchgeführt. Sowohl beim Vermessen des Randes als auch beim Vermessen des Inneren des Blattes kann man den Messvorgang beliebig viele Male unterbrechen und an einem anderen Punkt fortsetzen. Die entscheidende Information für das Zeichnen des Blattes ist die Anzahl der Vertices, die zum Rand gehören. Die Information wird zur Erkennung der Blattstruktur in der triangulierten Punktwolke benutzt. Die Triangulierung der aufgenommenen Punkte wird mit der Funktion Delaunay des Python-Moduls scipy.spatial (The Scipy community 2012) erstellt. Der Algorithmus, der in der Bibliothek für die Triangulierung benutzt wird, kann sowohl in 2D als auch in 3D eingesetzt werden und beruht auf der Delaunay-Triangulation. Dabei werden die Datenpunkte in eine Menge von zusammenhängenden Dreiecken umgewandelt. Die Vereinigung der Dreiecke bildet eine konvexe Hülle (Farin und Hansford 2003) (Abb. 24).

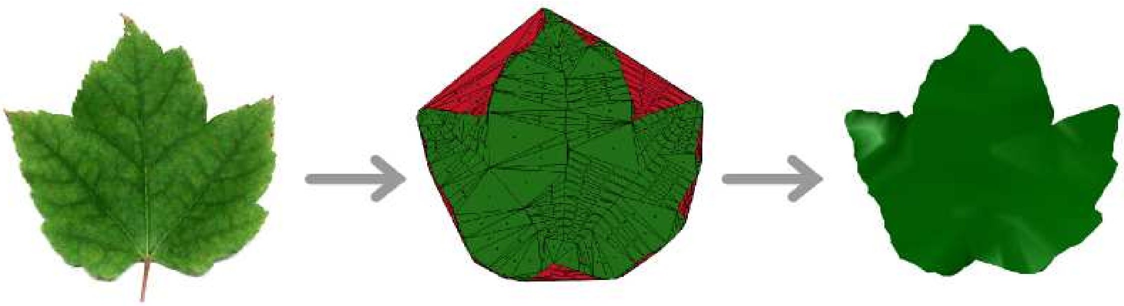


Abb. 24: Der Übergang von einem realen Blatt über die in der Delaunay-Triangulation entstandene konvexe Hülle zu der Darstellung im Rechner. In dem mittleren Bild sind die überflüssigen Dreiecke rot markiert.

Die 3D-Punktwolke aus dem gescannten Blatt wird zuerst auf die XY-Ebene projiziert, und erst danach wird die Delaunay-Funktion auf den 2D Daten ausgeführt. Für die Projektion wird von jedem Vertex der aufgenommenen Punktwolke der Schwerpunkt o der Punktwolke abgezogen. In Blender wird der Schwerpunkt der Geometrie bei jedem Objekt in dem Attribut location

abgespeichert und kann jederzeit abgefragt werden. Nachdem die lokalen Koordinaten d_i der Punkte p_i ausgerechnet wurden,

$$d_i = p_i - o$$

können die Normalenvektoren n_i für zwei aufeinander folgenden Randpunkte d_i und d_{i+1}, die zusammen mit dem Schwerpunkt und der zugehörigen Randkante einen Dreieck definieren, ausgerechnet werden

$$n_i = \frac{d_i \times d_{i+1}}{|d_i||d_{i+1}|}.$$

Als nächstes wird der Mittelwert der berechneten Normalenvektoren ermittelt und normiert:

$$\bar{n} = \frac{1}{N}\sum_{i=1}^{N} n_i,$$

$$\bar{n} = \frac{\bar{n}}{|\bar{n}|}.$$

Der normierte Mittelwert der Normalenvektoren dient zuerst zum Testen, ob für die Projektion der Punktwolke auf die XY-Ebene zuerst eine Rotation durchgeführt werden soll. Dies wird angenommen, wenn der Betrag der z-Koordinate des mittleren Normalenvektors kleiner 0.8 ist. Die Rotationsmatrix wird hier mit der Funktion Matrix.Rotation des mathutils-Moduls von Blender aufgestellt. Die Drehachse ist durch die y- und x-Koordinate des mittleren Normalenvektors (av_n) und der Winkel über die z-Koordinate, die den Kosinus des Winkels enthält, bestimmt.

```
# the function arguments are: angle, size of the matrix (4x4), rotation
axis
rot_mat = Matrix.Rotation(-arccos(av_n[2]),4,Vector((av_n[1],-av_n[0], 0)))
```

Die Punkte für die 2D-Triangulation t_i werden aus dem Anteil in der Normalenrichtung und den lokalen d_i Vektoren gewonnen.

$$t_i = d_i - (d_i \cdot \bar{n})\bar{n}$$

Bei Bedarf werden die Vektoren t_i mit der Drehmatrix multipliziert. Danach kann die z-Koordinate weggelassen werden, und die 2D-Triangulation mit der Delaunay-Funktion liefert eine Liste der Polygone. In jedem Eintrag der Liste werden jeweils drei Indizes der Vertices der Punktwolke abgespeichert. Der nächste Schritt ist das Herausfiltern der Dreiecke, die außerhalb des Blattumfangs gezeichnet würden (Abb. 24). Das Testverfahren wird in einer for-Schleife durchgeführt, die über alle Einträge der Polygonliste aus der Delaunay-Triangulation iteriert. Zuerst wird getestet, ob einer der Indices zu einem Punkt aus dem Inneren des Blattes gehört. Wenn das der Fall ist, wird dieses Polygon sofort gezeichnet. Wenn die Bedingung nicht zutrifft, wird in den 2D-Daten getestet, ob das Dreieck außerhalb der Kontur des Blattes liegt oder nicht. Dies geschieht durch die Berechnung des Kreuzproduktes zwischen dem ersten und dem zweiten Kantenvektor des Polygons.

Wenn die z-Koordinate des Ergebnisses negativ ist, liegt das zugehörige Dreieck im Inneren des Blattes und kann gezeichnet werden. Damit der Außen-Innen-Test durchgeführt werden kann, müssen die Randpunkte gegen den Uhrzeigersinn vermessen werden. Die Messung der Randpunkte im Uhrzeigersinn innerhalb der Kontur führt zur Entstehung eines Lochs. Wenn das Zeichnen abgeschlossen ist, werden die Normalenvektoren der Polygone mit unify_normals aktualisiert, so dass sie sich alle auf einer Seite des Blattes befinden (Abb. 25). Die Normalenvektoren werden in Blender unter anderem für richtige Darstellung des Lichts und Schattens gebraucht.

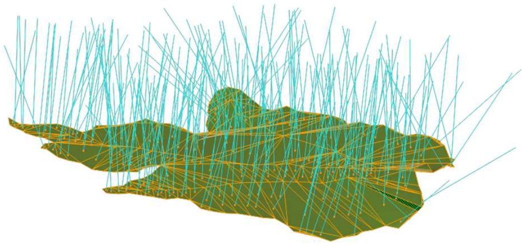


Abb. 25: Die aktualisierten Normalenvektoren der Polygone eines Blattes.

Zum Schluss wird für ein Blatt dessen Fläche aus den Flächen der gezeichneten Polygone ermittelt.

```
# calculate the area of the object
for poly in bpy.context.activ_object.data.faces:
    area = area + poly.area
# update the area property of the object
bpy.context.activ_object.fk_o.area = area
```

Die dritte optionale Form, die man in dem kontinuierlichen Modus des Scanners bei der Aufnahme der Topologie erstellen kann, ist eine 3D-Geometrie, die zur Darstellung der gemessenen Früchte dient. Hierfür wird genauso wie bei der Erstellung der Blätter und Blüten die Delaunay-Triangulation verwendet. Diesmal werden aber der Delaunay-Funktion des scipy.spatial-Moduls die 3D-Punkte übergeben. Auch hier wird zuerst eine konvexe Hülle aus den Dreiecken erstellt, die dann für die Darstellung in Blender nachbearbeitet werden muss (Abb. 26).

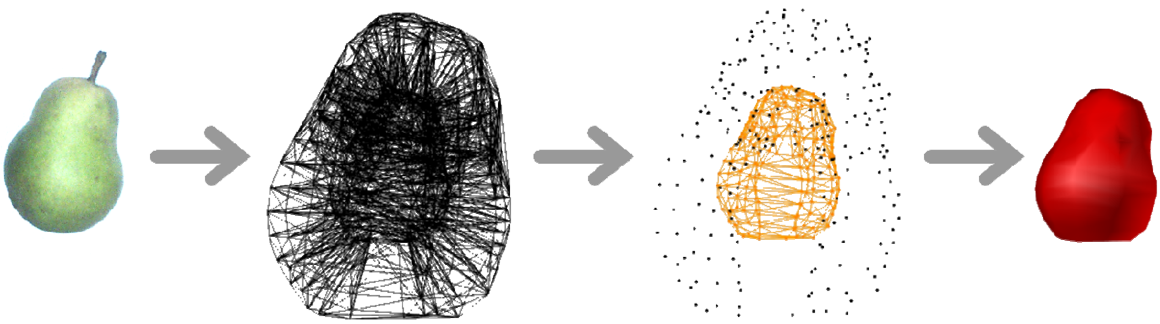


Abb. 26: Der Weg von einer realen Frucht zu einem 3D-Modell im Rechner. In der Abbildung links befindet sich die vermessene Birne. Als nächstes wird bei der Delaunay-Triangulation die konvexe Hülle erstellt. Danach erfolgt das Filtern der Oberflächendreiecke, und anschließend wird das Modell in dem 3D-Fenster gezeichnet.

Die 3D-Objekte werden nach der Aufnahme der Datenpunkte in drei Phasen nachbearbeitet. Bei der ersten Phase werden die Scanpunkte dupliziert und ihre Ortsvektoren skaliert, so dass sie sich um das Objekt herum befinden. Dies wird in der dritten Phase benutzt, um zu entscheiden, ob ein Dreieck aus einem Tetraeder in das Modell übernommen werden soll. In der zweiten Phase wird mit der Delaunay-Funktion des scipy.spatial-Moduls eine Triangulation der 3D-Punkte durchgeführt. Die Triangulation einer 3D-Punktwolke ist in der Liste mit den Vertices der Tetraeder abgespeichert. Jeder Tetraedereintrag in der Liste beinhaltet vier Indizes der zugehörigen Vertices. Die Daten werden in der dritten Phase auf die Zugehörigkeit zu der Oberfläche des gescannten Objekts getestet. Dank der zusätzlichen Punkte kann man leicht erkennen, welche Dreiecke zu dem Modell gehören sollen. Es werden nur die Dreiecke zum Zeichnen zugelassen, die sich in einem Tetraeder befinden, der nur drei echte, gescannte Vertices besitzt. Alle andere Tetraeder und deren Vertices werden ignoriert.

```
faces = []
for tetraeder in triangulation.tetraeder:
    triangle = []
    # search for triangles
    for vertex_index in tetraeder:
        if vertex_index < real_vertices_number:
            triangle.append(vertex_index)
    # create the object faces list
    if len(triangle) == 3:
        faces.append(triangle)
```

Dieses Verfahren funktioniert in den meisten Fällen. Die Voraussetzung für die richtige Erkennung der Oberfläche ist, dass die duplizierten Punkte nach der Skalierung wirklich um das gescannte Objekt herum platziert sind. Diese Beschränkung zu beheben, ist für eine zukünftige Version des Addons vorgesehen. Nachdem die dritte Phase der Nacharbeitung der Punkte abgeschlossen wurde, werden die ausgewählten Dreiecke zu dem gescannten Objekt hinzugefügt und die Duplikate der Vertices entfernt.

Zeichnen der Geometrie für die Berechnung des Volumens

Das Zeichnen der für Berechnung des Volumens und der Masse des gescannten Gegenstands nötigen Geometrie wird genauso wie in Falle eines Objektes durchgeführt. Der Unterschied besteht hier darin, dass die Objekte nicht nach dem MTG-Format benannt werden und kein Material zugewiesen bekommen. Sie besitzen stattdessen eine Information über das Volumen des realen Objekts und, falls der Wert der Dichte angegeben ist, die Information über die Masse des gescannten Objekts. Die Berechnung des Volumens kann mit dem Operator SCENE_OT_fk_obj_setup_volume in dem Panel mit Objektinformationen berechnet werden. Die Beschreibung der Methode folgt in einem weiteren Abschnitt des Kapitels (siehe Seite 99).

Das Modul f_operators.py

In diesem Modul befinden sich fast alle Operatoren, die in dem Addon gebraucht werden. Einige wenige, die bei der seriellen Kommunikation helfen, sind in dem Modul __init.py enthalten (siehe Seite 51). Die Operatoren sind in Blender der essentielle Teil der Benutzerschnittstelle der Anwendung. Sie stellen ihm die Funktionalitäten des Programms zur Verfügung. Jeder Operator kann in der Benutzeroberfläche als ein Button platziert und benutzt werden (siehe Seite 27). Da die Operatoren in dem Addon relativ einfach aufgebaut sind und meistens sich schon aus dem Namen die Funktionalität ablesen lässt, werden nur die Operatoren besprochen, bei deren Implementierung mehr theoretisches oder technisches Hintergrundwissen gebraucht wird. Alle in dem Modul definierten Operatoren sind in Tab. 15 aufgelistet.

Tab. 15: Operatoren des Moduls fk_operators.py**.**

Platzierung im UI	Name der Klasse	Funktionalität
Kontrollpanel	FASTRAK_contr_export_csv	Exportiert die Scandaten im CSV-Format.
	FASTRAK_contr_export_mtg	Exportiert die Scandaten im MTG-Format.
	SCENE_OT_fk_contr_info_show_names	Zeigt die Namen der selektierten Objekte im 3D-Fenster an.
	SCENE_OT_fk_contr_hide_names	Versteckt die Namen der selektierten Objekte im 3D-Fenster.
Adjustment Panel *(Aufnehmen der Geometrie für die Kalibrierung)*	SCENE_OT_fk_adj_cal_accept_shift	Akzeptiert die Werte der Verschiebung der Kalibierungsbox im Bezug auf den Sender. Aktualisiert die UI-Eigenschaften, so dass die nächste Ansicht angezeigt werden kann.
	SCENE_OT_fk_adj_cal_request_point	Fordert einen Punkt vom Scanner an.
	SCENE_OT_fk_adj_cal_pop_point	Löscht den letzten dargestellten Punkt.
	SCENE_OT_fk_adj_cal_finish	Aktualisiert die UI-Eigenschaften, so dass die nächste Ansicht angezeigt werden kann.
	SCENE_OT_fk_adj_cal_clear	Löscht die Kalibrierungsgeometrie und erlaubt eine neue aufzunehmen.
	SCENE_OT_fk_adj_calc_coef_mat	Berechnet die Koeffizientenmatrix aus den abgetasteten Daten.

Adjustment Panel *(Setzen der Referenzpunkte und Berechnung der Entzerrungsmatrix)*	SCENE_OT_fk_adj_ref_request_point	Fordert einen Punkt vom Scanner an.
	SCENE_OT_fk_adj_rm_ref	Löscht den letzten aufgenommenen Punkt aus der gewählten Referenzgruppe.
	SCENE_OT_fk_adj_calc_equalization	Berechnet die Entzerrungsmatrix für die Korrektur der neu gescannten Daten.
Alignment Panel *(Setzen der Nord-Süd Ausrichtung)*	SCENE_OT_fk_al_geo_point	Fordert einen Punkt vom Scanner an.
	SCENE_OT_fk_al_geo_clear	Löscht die Objekte für die Markierung der geographischen Richtung. Aktualisiert die UI-Eigenschaften, so dass die nächste Ansicht angezeigt werden kann.
Alignment Panel *(Aufnehmen der Punkte für die Berechnung der Neigung zu der XY-Ebene des Senders)*	SCENE_OT_fk_al_slope_start	Startet die Aufnahme der Daten für die Berechnung der Neigung. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die nächste Aktion vorbereitet ist.
	SCENE_OT_fk_al_slope_stop	Stoppt die Aufnahme der Daten für die Berechnung der Neigung. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die nächste Aktion vorbereitet ist.
	SCENE_OT_fk_al_slope_finish	Schließt das Zeichnen des Objektes ab. Aktualisiert die UI-Eigenschaften, so dass die nächste Ansicht angezeigt werden kann.
	SCENE_OT_fk_al_slope_clear	Löscht das angezeichnete Objekt. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die Aufnahme von neuen Daten vorbereitet ist.
	SCENE_OT_fk_al_calc_slope	Berechnet aus den aufgenommenen Daten die Neigung gegen die XY-Ebene des Senders.
Topologie Panel *(Darstellung der vermessen Objekte mit einfachen Geometrieformen – Sphäre, Linie, Zylinder)*	SCENE_OT_fk_topo_new_axis	Fügt eine neue Achse (eine Blender-Gruppe) ein.
	SCENE_OT_fk_topo_pop_axis	Löscht die zuletzt erstellte Gruppe zusammen mit allen zugehörigen Objekten aus der Blender-Gruppenliste.
	SCENE_OT_fk_topo_request_point	Fordert einen Punkt vom Scanner an.
	SCENE_OT_fk_topo_delete_last_obj	Löscht das letzte bei dem Scan der Standardtopologie der Pflanze aufgezeichnete Objekt.
Topologie Panel *(Darstellung der vermessen Objekte mit freien Geometrieformen – Blatt)*	SCENE_OT_fk_topo_leaf_border_start	Startet die Aufnahme des Randes eines Blattes. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die Aufnahme von neuen Daten vorbereitet ist.
	SCENE_OT_fk_topo_leaf_border_stop	Stoppt die Aufnahme der Daten für das Zeichen des Randes. Aktualisiert die UI-Eigenschaften, so dass der nächste Schritt möglich ist.

	SCENE_OT_fk_topo_leaf_border_finish	Schließt die Aufnahme des Blattrandes ab und aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die Aufnahme von neuen Daten des Blattinneren vorbereitet ist.
	SCENE_OT_fk_topo_leaf_inside_start	Startet die Aufnahme des Inneren eines Blattes. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die Aufnahme von neuen Daten vorbereitet ist.
	SCENE_OT_fk_topo_leaf_inside_stop	Hält die Aufnahme der Daten für das Zeichen des Blattinneres an. Aktualisiert die UI-Eigenschaften, so dass der nächste Schritt möglich ist.
	SCENE_OT_fk_topo_leaf_inside_finish	Schließt die Aufnahme des Blattinneren ab. Aktualisiert die UI-Eigenschaften, so dass das komplette Blatt gezeichnet wird und die Benutzeroberfläche für die Aufnahme von neuen Blattdaten vorbereitet ist.
	SCENE_OT_fk_topo_leaf_clear_draw	Löscht das gezeichnete Blattobjekt und aktualisiert die UI-Eigenschaften, so dass das Blatt neu gezeichnet werden kann.
Topologie Panel *(Darstellung der vermessen Objekte mit freien Geometrieformen – Blüte)*	SCENE_OT_fk_topo_flower_border_start	Startet die Aufnahme des Randes einer Blüte. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die Aufnahme von neuen Daten vorbereitet ist.
	SCENE_OT_fk_topo_flower_border_stop	Hält die Aufnahme der Daten für das Zeichen des Randes an. Aktualisiert die UI-Eigenschaften, so dass der nächste Schritt möglich ist.
	SCENE_OT_fk_topo_flower_border_finish	Schließt die Aufnahme der Blütenkontur ab und aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die Aufnahme von neuen Daten des Blüteinneren vorbereitet ist.
	SCENE_OT_fk_topo_flower_inside_start	Startet die Aufnahme des Inneren einer Blüte. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die Aufnahme von neuen Daten vorbereitet ist.
	SCENE_OT_fk_topo_flower_inside_stop	Hält die Aufnahme der Daten für das Zeichnen des Blüteinneren an. Aktualisiert die UI-Eigenschaften, so dass der nächste Schritt möglich ist.
	SCENE_OT_fk_topo_flower_inside_finish	Schließt die Aufnahme des Blüteinneren ab. Aktualisiert die UI-Eigenschaften, so dass die komplette Blüte gezeichnet wird und die Benutzeroberfläche für die Aufnahme von neuen Blütendaten vorbereitet ist.
	SCENE_OT_fk_topo_flower_clear_draw	Löscht das gezeichnete Blütenobjekt und aktualisiert die UI-Eigenschaften, so dass die Blüte neu gezeichnet werden kann.
Topologie Panel *(Darstellung der vermessen Objekte mit freien Geometrieformen – Frucht)*	SCENE_OT_fk_topo_fruit_surf_start	Startet die Aufnahme der Datenpunkte für das Zeichnen einer Frucht. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für die Aufnahme von neuen Daten vorbereitet ist.

	SCENE_OT_fk_topo_fruit_surf_stop	Hält die Aufnahme der Daten für das Zeichen einer Frucht an. Aktualisiert die UI-Eigenschaften, so dass der nächste Schritt möglich ist.
	SCENE_OT_fk_topo_fruit_surf_finish	Schließt die Aufnahme der Fruchtdaten ab. Aktualisiert die UI-Eigenschaften, so dass die komplette Fruchtgeometrie gezeichnet wird und die Benutzeroberfläche für die Aufnahme von neuen Blütendaten vorbereitet ist.
	SCENE_OT_fk_topo_fruit_surf_clear	Löscht das gezeichnete Fruchtobjekt und aktualisiert die UI-Eigenschaften, so dass die Frucht neu gezeichnet werden kann.
Volume Panel	SCENE_OT_fk_vol_surf_start	Startet die Aufnahme der Datenpunkte für das Zeichnen einer Geometrie zur Berechnung des Volumens. Aktualisiert die UI-Eigenschaften, so dass die Benutzeroberfläche für Aufnahme von neuen Daten vorbereitet ist.
	SCENE_OT_fk_vol_surf_stop	Hält die Aufnahme der Daten für das Zeichen einer Geometrie zur Berechnung des Volumens an. Aktualisiert die UI-Eigenschaften, so dass der nächste Schritt möglich ist.
	SCENE_OT_fk_vol_surf_finish	Schließt die Aufnahme der Geometriedaten ab. Aktualisiert die UI-Eigenschaften, so dass das komplette Objekt gezeichnet, das Volumen berechnet wird und die Benutzeroberfläche für die Aufnahme von neuen Daten vorbereitet ist.
	SCENE_OT_fk_vol_surf_clear	Löscht die gezeichnete Geometrie und aktualisiert die UI-Eigenschaften, so dass ein neues Objekt vermessen werden kann.
MTG Panel *(Bifurcation)*	SCENE_OT_fk_mtg_set_basis	Setzt den Namen des aktiven Objekts in das Eingabefeld für die Verzweigungsbasis.
	SCENE_OT_fk_mtg_set_sucessor	Setzt den Namen des aktiven Objekts in das Eingabefeld für den ersten Nachfolger in der Verzweigung.
	SCENE_OT_fk_mtg_add_branch	Fügt eine neue Verzweigung in die Verzweigungsliste an. Aktualisiert die Eigenschaften der Benutzeroberfläche und der zugehörigen Objekte.
	SCENE_OT_fk_mtg_select_branch	Markiert alle Objekte, die zu der Verzweigungsgruppe gehören.
	SCENE_OT_fk_mtg_unlink_branch	Löscht die Verzweigung aus der Verzweigungsliste, ohne die zugehörigen Objekte zu löschen. Aktualisiert demensprechend die Eigenschaften der Benutzeroberfläche und der zugehörigen Objekte.
MTG Panel *(Wachstumseinheiten)*	SCENE_OT_fk_mtg_add_unit	Fügt eine neue Wachstumseinheitengruppe zu der Gruppenliste hinzu. Aktualisiert die Eigenschaften der Benutzeroberfläche und der zugehörigen Objekte.

	SCENE_OT_fk_mtg_pop_unit	Löscht die zuletzt hinzugefügte Wachstumseinheitengruppe, ohne die Objekte zu löschen. Aktualisiert demensprechend die Eigenschaften der Benutzeroberfläche und der zugehörigen Objekte.
	SCENE_OT_fk_mtg_select_unit	Selektiert alle Objekte, die zu der Wachstumseinheitengruppe gehören.
	SCENE_OT_fk_mtg_unit_link_nodes	Fügt die Objekte zu der Wachstumseinheitengruppe hinzu. Aktualisiert demensprechend die Eigenschaften der Benutzeroberfläche und der zugehörigen Objekte.
	SCENE_OT_fk_mtg_unit_unlink_nodes	Entfernt die Objekte aus der Wachstumseinheitengruppe, ohne dass sie aus dem Programm gelöscht werden.
MTG Panel *(Pflanzengruppen)*	SCENE_OT_fk_mtg_add_plant	Fügt eine neue Pflanzengruppe zu der Gruppenliste hinzu. Aktualisiert die Eigenschaften der Benutzeroberfläche und der zugehörigen Objekte.
	SCENE_OT_fk_mtg_pop_plant	Löscht die zuletzt hinzugefügte Pflanzengruppe, ohne die Objekte zu löschen. Aktualisiert demensprechend die Eigenschaften der Benutzeroberfläche und der zugehörigen Objekte.
	SCENE_OT_fk_mtg_select_plant	Selektiert alle Objekte, die zu der Pflanzengruppe gehören.
	SCENE_OT_fk_mtg_plant_link_nodes	Fügt die Objekte zu der Pflanzengruppe hinzu. Aktualisiert demensprechend die Eigenschaften der Benutzeroberfläche und der zugehörigen Objekte.
	SCENE_OT_fk_mtg_plant_unlink_nodes	Entfernt die Objekte aus der Pflanzengruppe, ohne dass sie aus dem Programm gelöscht werden.
Panel für Objektinformation	SCENE_OT_fk_o_info_split_diam	Aktualisiert die Eigenschaften der Benutzeroberfläche so, dass die Durchmesser eines Zylinders für den oberen und unteren Deckel getrennt angegeben werden können.
	SCENE_OT_fk_o_info_merge_diam	Aktualisiert die Eigenschaften der Benutzeroberfläche so, dass die Durchmesser eines Zylinders für den oberen und unteren Deckel gleichzeitig geändert werden.
	SCENE_OT_fk_o_info_setup_volume	Berechnet das Volumen des aktiven Objekts neu.

Löschen der Objekte

Bei der Vermessung ist es wichtig, dass der Benutzer eine Möglichkeit hat, die gezeichneten Objekte zu entfernen. Blender bietet einen Operator für das Entfernen der Objekte. Dabei werden keine Metadaten aus dem Scan aktualisiert. Der Operator löscht auch keine Gitterdaten, die dem Objekt gehörten. Deswegen gibt es im Addon für jeden Typ des gescannten Objekts einen speziellen Operator,

der für das Löschen eines Objekts, bzw. einer Gruppe von Objekten, sowie für die Aktualisierung der Addon-Daten und der Benutzeroberfläche verantwortlich ist.

```
# make sure that only the active object will be removed
deselect_objs(context)
bpy.context.active_object.select = True

# delete only object data without mesh data
bpy.ops.object.delete()

## meshes have no users anymore
bpy.data.meshes.remove(bpy.data.meshes[bpy.context.active_object.name])
```

Export der Modelldaten aus Blender

Der Export der gescannten Daten wird mit Hilfe von zwei Operatoren durchgeführt. Mit dem Operator FASTRAK_export_csv werden alle Informationen zu den selektierten Objekten in eine csv-Datei geschrieben. Mit FASTRAK_export_mtg-Operator ist es möglich, die Daten der selektierten Objekte, die aus dem Scan der topologischen Struktur der Pflanze entstammen, in eine MTG-Datei zu schreiben. Der Operator kann zurzeit den MTG-Graphen nur auf der Ebene der Internodien exportieren. Die zusätzlichen Informationen, die im MTG-Editor den Objekten zugeweisen werden, kann man nur mit dem CSV-Export wegschreiben. Desweiteren sind in der exportierten MTG-Datei, falls vorhanden, die Informationen über die Länge, den Durchmesser und den Typ des Internodiums enthalten.

Die Implementierung der Export-Operatoren wurde anhand der Blender-Schablonen implementiert. Die Exportoperatoren unterscheiden sich von normalen Operatoren zuerst dadurch, dass sie von der Klasse ExportHelper erben. Im Rumpf von solchen Operatoren muss die Erweiterung der zu exportierenden Datei und der Filter für die Anzeige der Dateien angegeben werden.

```
# inheritance of two classes
class Export_operator(Operator, ExportHelper):
    # ...
    # file extension for export
    filename_ext = ".csv"
    # the filter for display of the files
    filter_glob = StringProperty(default = "*.csv", options = {'HIDDEN'},)
```

Die erste Funktion der Klasse, die nach dem Anklicken des Buttons aufgerufen wird, ist invoke. Sie öffnet eine neue Blender-Ansicht, die der Selektion und dem Abspeichern der Dateien dient. Die Ansicht wird solange angezeigt, bis der Benutzer den Button Cancel drückt oder die Datei unter dem angegebenen Namen abspeichert.

```
# open the export view
def invoke(self, context, event):
    context.window_manager.fileselect_add(self)
    return {'RUNNING_MODAL'}
```

Wenn der Benutzer die Datei speichert, wird die execute-Funktion der Klasse aufgerufen. In dieser Funktion werden die Informationen in der angegebenen Lokalisation abgespeichert.

```
# write informations to the file
def execute(self, context):
    f = open(self.filepath, 'w', encoding='utf-8')
    f.write("export data goes here")
    f.close()
    return {'FINISHED'}
```

Damit der selbst geschriebene Exportoperator in dem File-Menu von Blender angezeigt werden soll, muss man die Funktion, am besten im __init.py-Modul des Addons, definieren und in der register-Funktion des Moduls mit bpy.types.INFO_MT_file_export.append zum Menü hinzufügen.

```
# define bind function to file menu
def menu_func_export(self, context):
    self.layout.operator(Export_operator.bl_idname, text = " Export(.csv)")

# bind the export operator to file menu in the function register
    bpy.types.INFO_MT_file_export.append(menu_func_export)
```

Berechnung der Entzerrungsmatrix

Der Benutzer soll eine Möglichkeit haben, die Vermessung nach Änderung der Position des Senders fortzusetzen. Für die Anpassung des neuen Koordinatensystems an das alte muss eine Entzerrungsmatrix X berechnet werden. Sie wird im Operator SCENE_OT_fk_adj_calc_equalization berechnet. Geometrisch betrachtet, setzt sich die Entzerrungsmatrix aus der Hintereinanderausführung der Matrizen für Skalierung (S), Rotation (R) und Translation (T) (Nischwitz, Fischer und Haberäcker 2007) zusammen.

$$X = SRT$$

Die berechnete Entzerrungsmatrix wird, falls es gewünscht ist, bei jedem ankommenden Punkt zur Korrektur der Koordinaten verwendet. Die Daten für die Berechnung der Matrix werden über das Panel FASTRAK: Adjustment/Refereces gesammelt (siehe Seite 124). Die Entzerrungsmatrix wird aus vier Punkten, die vor der Verschiebung des Senders abgemessen wurden (4x4-Matrix „before“ := B), und vier Punkten, die nach der Verschiebung des Senders gemessen wurden (4x4-Matrix „after“ := A), berechnet. Die Punkte werden spaltenweise in die Matrizen A und B geschrieben. Die Entzerrungsmatrix ist die Lösung von:

$$XA = B$$

$$X = BA^{-1}$$

Sowohl die Entzerrungsmatrix als auch die Datenpunkte werden in homogenen Koordinaten in die Berechnung aufgenommen. Nachdem die Entzerrungsmatrix berechnet wurde, wird sie in der Eigenschaft bpy.context.scene.fk_adj.eq_mat als ein Vektor abgespeichert. Wenn die Eigenschaft gesetzt ist, kann man über das Kontrollpanel die Korrektur für die neugemessenen Punkte einschalten. Bei der Vermessung wird dann online in der Funktion __prepare_float_line der Klasse

SCENE_OT_fk_scan_manager des __init.py-Moduls das folgende Gleichungssystem gelöst.

$$Xp' = p$$

Dabei steht p für den gemessenen Datenpunkt nach der Verschiebung des Senders und p′ für das zu zeichnende Ergebnis der Korrektur.

Berechnung der Kalibrierungsmatrix

Die Kalibrierungsmethode für die Position der gescannten Punkte wurde anhand von zwei Artikeln (Ikits, et al. 2001) (Kindratenko 1999) in das Addon eingebaut (SCENE_OT_fk_adj_calc_coef_mat). Für die Erklärung wurden aus den Artikeln die für die Implementierung relevanten Teile entnommen und zusammengestellt.

Eine häufig zur Kalibrierung des magnetischen Felds benutzte Methode ist die Korrektur per polynomialem Fit. Der Vorteil dieser Methode ist deren Robustheit, Einfachheit und Schnelligkeit. Man braucht nur wenige Parameter, um den Fehler an einer gegebenen Position zu berechnen. Nachteilig ist, dass sie keine gute Anpassung für kleine Fehler nah am Sender liefert und nicht genau genug für die Beschreibung der lokalen Feldverzerrung ist. Die polynomialen Koeffizienten können z.B. mit der Methode der kleinsten Quadrate berechnet werden. Dabei wird die Differenz zwischen dem gemessenen Fehler und den Werten aus dem Kalibrierungsmodell minimiert.

Das Polynom r-ten Grades für den Positionsvektor p kann wie folgt definiert werden:

$$f(c,p) = f(c,x,y,z) \ = \sum_{j=1}^{R} \begin{bmatrix} c_{x,j} \\ c_{y,j} \\ c_{z,j} \end{bmatrix} x^{s_j} y^{u_j} z^{w_j}.$$

Dabei ist jeder Term des Polynoms anders und höchstens von Grad r. R steht für die Anzahl der Terme im Polynom. In dem Polynom müssen alle Kombinationen der Potenzen, deren Exponentensummen in dem Intervall [0, r] liegen, vorhanden sein.

$$0 \le s_j + u_j + w_j \le r$$

Die Potenzen der Koordinaten x, y und z sind nicht negativ, und jede Potenzkombination muss dabei eindeutig sein. Unter dieser Bedingung ist die Anzahl der Terme

$$R = \frac{(r+1)(r+2)(r+3)}{6}.$$

Für die Durchführung der Korrektur muss ein Koeffizientenvektor c gefunden werden, für den das angepasste Polynom den Kalibrierungsfehler an den Messorten am besten approximiert. Dies lässt sich als ein Minimierungsproblem für das folgende Fehlerfunktional darstellen.

$$S_p = \sum_{i=1}^{N} \left\| p_e^i - f\left(c_p, p_t^i\right) \right\|^2$$

Dabei bezeichnet N der Anzahl der aufgenommenen Daten, $p_t = (x_{distorted}, y_{distorted}, z_{distorted})$ den Vektor der von dem Trackingsystem gemessenen Position und p_e den Fehlervektor, der durch die Differenz zwischen der echten Position der Kalibrierungsvorrichtung $p_m = (x_{real}, y_{real}, z_{real})$ und der vom Empfänger gemessenen Position entsteht

$$p_e = p_m - p_t.$$

Die Lösung des Ausgleichproblems kann in der Matrixschreibweise, hier nur für die x-Koordinate der Kalibrierungspunkte, als $Ac_\alpha = b_\alpha$ definiert werden. Der Index α steht für die Koordinaten x, y oder z. Ausgeschrieben sieht die Matrix wie folgt aus:

$$\begin{pmatrix} a_{1,1} \cdots a_{1,j} \cdots a_{1,R} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{i,1} \cdots a_{i,j} \cdots a_{i,R} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{N,1} \cdots a_{N,j} \cdots a_{N,R} \end{pmatrix} \begin{pmatrix} c_{\alpha,1} \\ \cdots \\ c_{\alpha,j} \\ \cdots \\ c_{\alpha,N} \end{pmatrix} = \begin{pmatrix} b_{\alpha,1} \\ \cdots \\ b_{\alpha,j} \\ \cdots \\ b_{\alpha,N} \end{pmatrix}$$

Die Einträge der Matrix A entsprechen den Termen des Polynoms ($a_{i,j} = x_i^{s_j}\, y_i^{u_j}\, z_i^{w_j}$). Die Einträge des Vektors b enthalten die Differenz zwischen der echten und der gemessenen Position des Kalibrierungspunktes ($b_{\alpha,j} = \alpha_{real\ j} - \alpha_{distorted\ j}$). Das überbestimmte Gleichungssystem $Ac_\alpha = b_\alpha$ kann nur dann minimiert werden, wenn der Anzahl der aufgenommenen Daten (N) größer als der Anzahl der Terme (R) in dem polynomialen Fit ist (Ikits, et al. 2001). Wenn die Koeffizienten bekannt sind, kann man die korrigierte x'-Koordinate des gemessenen Punkts wie folgt ermitteln (Kindratenko 1999).

$$x' = x + \sum_{j=1}^{R} c_{x,j}\, x^{s_j} y^{u_j} z^{w_j}$$

Für die Korrektur der y'- und z'-Koordinate muss das gleiche Verfahren eingesetzt werden. Wenn dies getan ist, kann jeder gemessene Punkt online korrigiert werden. Der Benutzer kann im Kontrollpanel entscheiden, ob die Kalibrierungskorrektur eingesetzt werden soll. Dabei muss man beachten, dass das Einschalten der Korrektur einen Einfluss auf das Darstellen der Daten in dem 3D-Fenster und die Geschwindigkeit der Interaktion mit Blender haben kann.

Berechnung der Neigung gegen die XY-Ebene

Nachdem die Daten für die Berechnung aufgenommen wurden, ist in Blender ein Objekt für die Berechnung der Neigung erstellt und aktiv gesetzt. Somit kann man in dem Operator über die Blender-Struktur `bpy.context.active_object` alle Informationen über das Objekt ablesen. Die Koordinaten der Punkte sind in den Gitterdaten des Objektes in der Struktur `bpy.context.active_object.data.vertices` abgespeichert. Um den Winkel gegen die XY-Ebene des Senderkoordinatensystems

zu berechnen, muss zuerst durch die Punkte eine angepasste Ebene ermittelt werden. Für die Bestimmung einer Ebene werden drei Punkte p_1, p_2 und p_3 gebraucht, die aus den gescannten Daten berechnet werden. Hierfür muss zuerst die am besten angepasste Ebene gefunden werden. Zu der Ebene haben alle Punkte aus der eingescannten Punktwolke einen minimalen Abstand. Diese Ebene wird mit Hilfe des Trägheitstensors bestimmt. Dabei wird die Eigenschaft ausgenutzt, dass die Ebene, die normal zum größten Eigenvektor liegt, gerade die gesuchte Ebene ist (Landau und Lifschitz 1990). Somit kann man die drei Punkte der Ebene mit folgenden Formeln bestimmen:

$$p_1 = o = \left(\frac{1}{n}\sum_{i=1}^{n} x_i\,,\quad \frac{1}{n}\sum_{i=1}^{n} y_i\,,\quad \frac{1}{n}\sum_{i=1}^{n} z_i\right),$$

$$p_2 = \left(\frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n} x_i},\quad \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i},\quad \frac{\sum_{i=1}^{n} x_i z_i}{\sum_{i=1}^{n} x_i}\right),$$

$$p_3 = \left(\frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} y_i},\quad \frac{\sum_{i=1}^{n} y_i^2}{\sum_{i=1}^{n} y_i},\quad \frac{\sum_{i=1}^{n} y_i z_i}{\sum_{i=1}^{n} y_i}\right).$$

Der erste Punkt, durch den die Ebene gezeichnet wird, entsteht aus dem arithmetischen Mittel der x-, y- und z-Koordinaten aller Punkte. Der Schwerpunkt des Objektes wird in Blender im Attribut location zu jedem Gitterobjekt abgespeichert und ist hier mit o abgekürzt.

Bei der Berechnung der neuen Punkte für die Ebene wird auch der mittlere Abstand d_1 der gescannten Punkte $v_{i,j}$ zum Schwerpunkt der Punktwolke o ausgerechnet.

$$for\ i = 1, \dots, n:\quad dv_i = \|o - v_i\| = \sqrt{\sum_{j=1}^{3}\left(o_j - v_{i,j}\right)^2}$$

$$d_1 = \frac{1}{n}\sum_{i=1}^{n} dv_i$$

Danach werden die zwei anderen Abstände zwischen den neuen Punkten der Ebene ausgerechnet.

$$d_2 = \|o - p_2\|$$

$$d_3 = \|o - p_3\|$$

Diese Informationen werden für die Visualisierung der Ebene in dem 3D-Fenster verwendet. Die Ebene soll viereckig und gut erkennbar sein. Deshalb werden zwei Skalierungsfaktoren ausgerechnet und die Punkte p_2 und p_3 damit aktualisiert.

$$p_2{}' = o + \frac{d_1}{d_2} p_2$$

$$p_3{}' = o + \frac{d_1}{d_3} p_3$$

Danach werden die neuen Punkte zu dem aktiven Objekt mit *add* hinzugefügt. Anschließend werden die vier neuen selektierten Vertices in ein Polygon mit *add_face* verbunden.

```
# reference to the mesh data of the object
obd = bpy.context.active_object.data

# creates four places for the new vertices and adds the vertices
obd.vertices.add(4)
obd.vertices[-1].co = p_2
obd.vertices[-2].co = p_2 * (-1)
obd.vertices[-3].co = p_3
obd.vertices[-4].co = p_3 * (-1)

# deselects all vertices of the mesh and then selects only four new
vertices
deselect_mesh()
obd.vertices[-1].select = True
obd.vertices[-2].select = True
obd.vertices[-3].select = True
obd.vertices[-4].select = True

# creates face between the selected vertices
add_face()
```

In Blender kann man die Normalenvektoren der Polygone aus den Gitterdaten abfragen. Für die Sicherheit wird zuerst der Operator für das Setzen der Normalenvektorenausrichtung aufgerufen. Danach kann der Normalenvektor des neu gezeichneten Polygons abgelesen werden.

```
bpy.ops.object.mode_set(mode = 'EDIT')
bpy.ops.mesh.normals_make_consistent(inside = False)
bpy.ops.object.mode_set(mode = 'OBJECT')

plain_normal = obd.faces[-1].normal
```

Wenn der Normalenvektor der Ebene bekannt ist, kann man den Cosinus des Winkels θ zwischen dem Normalenvektor der neuen Ebene n_p und dem Normalenvektor der XY-Ebene $n_{xy} = (0, 0, 1)$ ausrechnen. Nach der geometrischen Definition des Skalarprodukts erhält man folgende Formel für den $\cos\theta$:

$$n_p \cdot n_{xy} = |n_p||n_{xy}| \cos\sphericalangle(n_p, n_{xy}), \quad \cos\sphericalangle(n_p, n_{xy}) = \cos\theta$$

$$=> \cos\theta = \frac{n_p \cdot n_{xy}}{|n_p||n_{xy}|}$$

Mit Hilfe des Arkustangens wird der Winkel

$$\theta = arccos(\cos\theta)$$

zwischen den zwei Ebenen ausgerechnet. Falls der ausgerechnete Winkel größer als $\pi/2$ ist, wird eine Korrektur des Winkels durchgeführt. Anschließend wird der ausgerechnete Wert der *slope*-Eigenschaft des Objekts aktualisiert. Der Winkel wird hier in Grad angegeben.

```
# angle correction
while alpha > pi/2:
    alpha = alpha - pi/2

# update the object property
bpy.context.active_object.fk_o.slope = degrees(alpha)
```

Berechnung des Volumens

Das Volumen wird mit Hilfe des Spatprodukts (Abb. 27) ermittelt (Farin und Hansford 2003) (Merz 2001). In einer for-Schleife wird für jedes Dreieck des Gitters des Objekts die Determinante der Matrix berechnet, deren Spalten von den Verbindungsvektoren des Schwerpunkts (Punkt A in Abb. 27) mit den Eckpunkten des Dreiecks (Punkte B, C und D in Abb. 27) gebildet werden:

$$V = \left|\sum_{i=1}^{n} \frac{1}{6} \det(\overrightarrow{AB}_i, \overrightarrow{AC}_i, \overrightarrow{AD}_i)\right|.$$

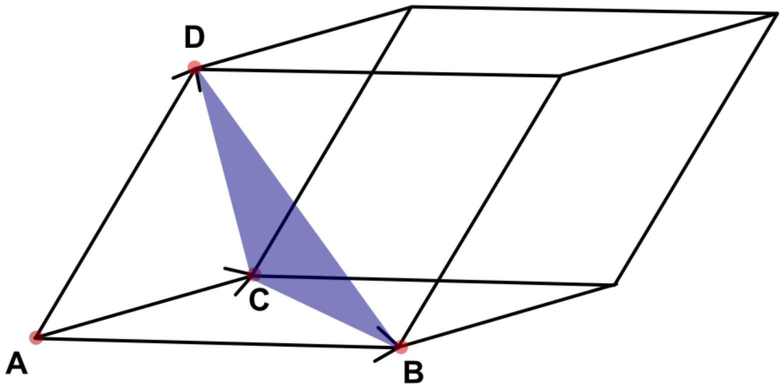


Abb. 27: Parallelepiped aufgebaut von drei Vektoren. Das blaue Dreieck ist ein Teil des Oberflächengitters des Objekts. A ist bei der Berechnung des Volumens der Objektschwerpunkt.

Bei der Ermittlung des Volumens mit der Methode müssen vier Bedingungen vor der Berechnung überprüft werden, damit das Ergebnis plausibel ist:

1. Alle Polygone des Objektes müssen aus drei Vertices aufgebaut sein.
2. Alle Normalen der Dreiecke müssen nach außen zeigen.
3. Das Gitter des Objektes darf keine Löcher haben.
4. Das Gitter des Objektes darf keine Dreiecke im Inneren des Objekts haben.

Wenn das Gitter vom Benutzer korrigiert wurde, kann man die Berechnung des Volumens neu starten und den alten Wert aktualisieren.

Das Modul f_properties.py

Das Modul f_properties.py dient der Definitionen der Klassen, die die Blender-Eigenschaften in dem Addon zur Verfügung stellen. Die Eigenschaften werden für

das Aufbauen und Steuern der Komponenten der Benutzeroberfläche, sowie zur Speicherung der gescannten Daten, Metainformationen zu den Objekten, Benutzereingaben und der Steuerzustände während des Scanverlaufs gebraucht. Da die Erstellung der Eigenschaften in Blender im ersten Kapitel (siehe Seite 23) behandelt wurde, wird in diesem Abschnitt nur eine Zusammenstellung aller in dem Addon definierten Eigenschaften mit einer kurzer Beschreibung in Tab. 16 angegeben.

Tab. 16: Eigenschaften des Addons für den Scan von Pflanzen.

Klasse	Eigenschaft	Blender-Typ	Beschreibung
SCENE_OT_fk_controller_pg	started	BoolProperty	Erscheint als Checkbox im Kontrollpanel. Dient zum Öffnen der Verbindung zum Scanner.
	ser_obj_ok	BoolProperty	Flag zur Information, ob die Verbindung mit dem Scanner möglich ist. Die Eigenschaft wird zur Steuerung des Anzeigens der Elemente der Oberfläche benutzt.
	increment	FloatProperty	Erscheint als Spinbox im Kontrollpanel. Speichert den Wert des Abstands der vom Scanner gesendeten Punkte.
	cursor_updates	IntProperty	Erscheint als Spinbox im Kontrollpanel. Speichert den Wert für die Aktualisierung des Cursors in dem 3D-Fenster.
	scan_type	EnumProperty	Erscheint als eine Combobox im Kontrollpanel. Aus der Liste kann man eines von den vier Panels für den Scan wählen.
	point_mode	BoolProperty	Enthält die Information, ob der Scan gerade im Einzelpunktmodus (1) oder kontinuierlichen Modus (0) durchgeführt wird.
	fButton_state	BoolVectorProperty	Enthält die Information über den letzten und aktuellen Status der Empfängertaste. Hilft bei der Verwaltung des Zeichnens der Objekte.
	draw_mesh	BoolProperty	Enthält die Information über den aktuellen Status des zu zeichnenden Gitters eines Objekts. Hilft bei der Verwaltung des Zeichnens der Objekte.
	use_calibr	EnumProperty	Wenn die Kalibrierungsmatrix berechnet ist, erscheint die Eigenschaft als eine Gruppe von Radiobuttons im Kontrollpanel. Mit Hilfe der Radiobuttons kann man die Kalibrierungskorrektur ein- und ausschalten.
	use_ref	EnumProperty	Wenn die Entzerrungsmatrix berechnet ist, erscheint die Eigenschaft als eine Gruppe von Radiobuttons im Kontrollpanel. Mit Hilfe der Radiobuttons kann man die Entzerrungskorrektur ein- und ausschalten.
SCENE_OT_fk_adjust_pg	type	EnumProperty	Erscheint im Adjustment-Panel als eine Gruppe von Radiobuttons. Ermöglicht das Umschalten zwischen Sammeln der Punkte für die Kalibrierung und Entzerrung.
	cal_shift	FloatVectorProperty	Erscheint im Adjustment-Panel als eine Gruppe von Spinbuttons für die Angabe der Entfernung der Kalibrierungsbox von dem Sender.
	cal_p_count	IntProperty	Zählt die aufgenommenen Kalibrierungspunkte. Wird als Information im Adjustment-Panel angezeigt.

	cal_ui_request	BoolProperty	Signal für das Anfordern eines Kalibrierungspunktes von der Software.
	cal_ui	BoolVectorProperty	Vektor für die Steuerung der Ansichten für nacheinander kommende Schritte der ersten Phase der Kalibrierung.
	calibr_ui	BoolVectorProperty	Vektor für die Steuerung der Ansichten für nacheinander kommende Schritte der zweiten Phase der Kalibrierung.
	calibr_mesh	BoolVectorProperty	Eigenschaft zur Kontrolle des Zeichnens der Kalibrierungsgeometrie.
	ref_groups	EnumProperty	Erscheint im Adjustment-Panel als eine Combobox. Dient der Wahl der Gruppe, zu der die vermessenen Punkte gehören sollen.
	ref_1_count	IntProperty	Zählt die aufgenommenen Referenzpunkte der ersten Gruppe. Wird als Information im Adjustment-Panel angezeigt.
	ref_2_count	IntProperty	Zählt die aufgenommenen Referenzpunkte der zweiten Gruppe auf. Wird als Information im Adjustment-Panel angezeigt.
	ref_ui_request	BoolProperty	Signal für das Anfordern eines Referenzpunktes von der Software.
	ref_mat_set	BoolProperty	Eigenschaft zur Steuerung der Anzeige der Radiobuttons für Ein- und Ausschalten der Referenzkorrektur im Kontrollpanel.
	equalisation_mat	FloatVectorProperty	Vektor zur Speicherung der Entzerrungsmatrix.
SCENE_OT_fk_alignment_pg	type	EnumProperty	Erscheint im Alignmet-Panel als eine Gruppe von Radiobuttons. Ermöglicht das Umschalten zwischen Sammeln der Punkte für das Setzen der geographischen Richtung und Berechnung der Neigung gegen die XY-Ebene des Senders.
	ui_request	BoolProperty	Signal für das Anfordern eines Punktes von der Software zum Setzen der geographischen Richtung.
	geo_exists	BoolVectorProperty	Information darüber, ob die Objekte für Nord und Süd schon gezeichnet worden sind. Falls ja, dürfen keine Objekte des Typs mehr gezeichnet werden.
	slope_ui	BoolVectorProperty	Vektor für die Steuerung der Ansichten der nacheinander kommenden Schritte der Aufnahme der Punkte für die Berechnung der Neigung.
	slope_mesh	BoolVectorProperty	Eigenschaft zur Kontrolle des Zeichnens des Objektes für die Berechnung der Neigung.
SCENE_OT_fk_topology_pg	type	EnumProperty	Erscheint im Topology-Panel als eine Gruppe von Radiobuttons. Ermöglicht das Umschalten zwischen Sammeln der Punkte für die die Digitalisierung der Standardtopologie und alternativer Formen von Pflanzenstrukturen.
	axes_counter	IntProperty	Zählt die im Scan der topologischen Struktur erstellten Achsen. Wird zur Information im Topology -Panel angezeigt.
	nodes_on_axis	IntProperty	Zählt die im Scan der topologischen Struktur erstellten Internodien, die zur letzten hinzugefügten Achse gehören. Wird zur Information im Topology -Panel angezeigt.

nodes_total	IntProperty	Zählt die im Scan der topologischen Struktur erstellten Internodien. Wird zur Information im Topology -Panel angezeigt.
p_old	FloatVectorProperty	Speichert die letzte aufgenommene Datenzeile. Wird beim Verketten der Linien und/oder Zylindern gebraucht.
ui_request	BoolProperty	Signal für das Anfordern eines Punktes von der Software zum Zeichnen der topologischen Struktur der Pflanze.
shape_type	EnumProperty	Erscheint im Topology-Panel als eine Combobox. Ermöglicht das Umschalten zwischen Zeichnen der Sphäre, Linie oder des Zylinders.
additional_nodes	EnumProperty	Erscheint im Topology-Panel als eine Combobox. Ermöglicht das Umschalten zwischen Zeichnen eines Blattes, einer Blüte oder Frucht in der Ansicht für die Aufnahme der zusätzlichen Strukturformen von Pflanzen.
inode_type	EnumProperty	Erscheint im Topology-Panel als eine Combobox. Ermöglicht das Umschalten zwischen Markierung eines Internodiums als ein Blatt, eine Blüte oder Frucht in der Ansicht für das Zeichnen der Standardformen für die Struktur von Pflanzen.
diameter	FloatProperty	Enthält die Information über den Durchmesser des zu zeichnenden Zylinders oder der zu zeichnenden Sphäre.
shp_n_seg	IntProperty	Enthält die Information über die Anzahl der vertikalen Segmente des zu zeichnenden Zylinders oder der zu zeichnenden Sphäre.
shp_n_rings	IntProperty	Enthält die Information über die Anzahl der horizontalen Segmente der zu zeichnenden Sphäre.
lf_bi	BoolVectorProperty	Vektor für die Steuerung der Ansichten der Benutzeroberfläche in den einzelnen Schritten des Abtastens der Kontur und des Inneren eines Blattes.
lf_mesh	BoolVectorProperty	Eigenschaft zur Kontrolle des Zeichnens eines Blattobjektes.
new_lf	BoolProperty	Signal, dass ein neues Blatt gezeichnet werden soll.
fl_bi	BoolVectorProperty	Vektor für die Steuerung der Ansichten der Benutzeroberfläche in den einzelnen Schritten des Abtastens der Kontur und des Inneren einer Blüte.
fl_mesh	BoolVectorProperty	Eigenschaft zur Kontrolle des Zeichnens eines Blütenobjektes.
fr_surf	BoolVectorProperty	Vektor für die Steuerung der Ansichten der Benutzeroberfläche in den einzelnen Schritten des Abtastens der Oberfläche einer Frucht.
fr_mesh	BoolVectorProperty	Eigenschaft zur Kontrolle des Zeichnens eines Fruchtobjektes.
info_2nd	BoolVectorProperty	Vektor zur Erkennung, wann während der Aufnahme der Punkte einer Form (Zylinder, Linie) aus zwei neuen Punkten gezeichnet werden soll und wann nur ein Punkt dafür gebraucht wird.

SCENE_OT_fk_volume_pg	v_surf	BoolVectorProperty	Vektor für die Steuerung der Ansichten der Benutzeroberfläche in den einzelnen Schritten des Abtastens der Oberfläche eines Objektes zur Ermittlung des Volumens dieses Objektes.
	v_mesh	BoolVectorProperty	Eigenschaft zur Kontrolle des Zeichnens der abgetasteten Form des Objektes zur Ermittlung des Volumens.
SCENE_OT_fk_mtg_pg	mtg_enum	EnumProperty	Erscheint im MTG Editor-Panel als eine Gruppe von Radiobuttons. Ermöglicht das Umschalten zwischen der Bearbeitung der Verzweigungs-, Wachstumseinheiten- und Pflanzengruppen.
	basis_node	StringProperty	Erscheint im MTG Editor-Panel als ein Eingabefeld für das Setzen des Namens des Basisinternodiums in der Verzweigung.
	succ_axis	StringProperty	Erscheint im MTG Editor-Panel als ein Eingabefeld für das Setzen des Namens für den ersten Nachfolger des Basisinternodiums.
	branch_ui	BoolVectorProperty	Vektor für die Steuerung der Ansichten der Benutzeroberfläche in den einzelnen Schritten des setzen einer Verzweigung.
	b_count	IntProperty	Zählt die Anzahl der gesetzten Verzweigungen auf. Wird in der Benutzeroberfläche zur Information eingeblendet.
	branch_name	StringProperty	Wird zur Erstellung eines neuen Eintrags in der Liste der Verzweigungen benutzt und ist in der Benutzeroberfläche zur Information über die neu gesetzte Verzweigung eingeblendet.
	u_count	IntProperty	Zählt die Anzahl der erstellten Gruppen der Wachstumseinheiten auf. Wird in der Benutzeroberfläche zur Information eingeblendet.
	unit_name	StringProperty	Wird zur Erstellung eines neuen Eintrags in der Liste der Wachstumseinheiten benutzt.
	activ_u_members	IntProperty	Wird in der Benutzeroberfläche zur Information über der Anzahl der Internodien, die zu der selektierten Wachstumseinheitengruppe gehören, eingeblendet.
	p_count	IntProperty	Zählt die Anzahl der erstellten Gruppen der Pflanzen auf. Wird in der Benutzeroberfläche zur Information eingeblendet.
	plant_name	StringProperty	Wird zur Erstellung eines neuen Eintrags in der Liste der Pflanzengruppen benutzt.
	activ_p_members	IntProperty	Wird in der Benutzeroberfläche zur Information über der Anzahl der Internodien, die zu der selektierten Pflanzengruppe gehören, eingeblendet.
OBJECT_OT_fk_object_prop	from_fk	BoolProperty	Ermöglicht zu unterscheiden, ob das Objekt mit Hilfe des Addons für den Scan von Pflanzen erstellt wurde (1) oder ein gwöhnliches Blender-Objekt ist (0). Alle in dem Scan erstellten Objekte besitzen diese Information.
	type	IntProperty	Speichert Informationen über den Typ des Objektes (siehe Tab. 11). Alle in dem Scan erstellten Objekte besitzen diese Information.
	timestamp	StringProperty	Speichert das Datum und die Zeit der Aufnahme der Geometrie. Alle in dem Scan erstellten Objekte besitzen diese Information.

slope	FloatProperty	Speichert das Ergebnis der Berechnung der Neigung gegen die XY-Ebene des Senders. Die Information ist für Objekte, die zu der Gruppe Slope gehören, reserviert und nur für diese Objekte wird sie im Objekt Info-Panel angezeigt.
other_gr	StringProperty	In dieser Eigenschaft wird der Name der Gruppe des Objekts abgespeichert. Dies gilt nur für die Objekte die nicht mit Hilfe des Topology-Panels aufgenommen wurden.
p1	FloatVectorProperty	Dieser Vektor speichert die sechs Freiheitsgrade, die für einen Punkt in dem Messraum vom Scanner ermittelt, und zum Zeichnen des Objekts benutzt wurden. Dies wird für Objekte im Einzelpunktmodus verwendet.
p2	FloatVectorProperty	Dieser Vektor speichert die sechs Freiheitsgrade, die für einen Punkt in dem Messraum vom Scanner ermittelt, und zum Zeichnen des Objekts benutzt wurden. Dies wird für Objekte in Einzelpunktmodus verwendet.
diameter	FloatProperty	Diese Eigenschaft speichert den Wert für den Durchmesser einer Sphäre, die im Topologiescan gezeichnet wurde.
cyl_d0	FloatProperty	Diese Eigenschaft speichert den Wert für den Durchmesser eines Zylinders, der im Topologiescan gezeichnet wurde. Der Durchmesser ist für den Deckel und den Boden des Zylinders gleich.
cyl_d1	FloatProperty	Diese Eigenschaft speichert den Wert für den Durchmesser des Bodens eines Zylinders, der im Topologiescan gezeichnet wurde. Sie wird als eine Spinbox im Objekt Info-Panel angezeigt und kann manipuliert werden.
cyl_d2	FloatProperty	Diese Eigenschaft speichert den Wert für den Durchmesser des Deckels eines Zylinders, der im Topologiescan gezeichnet wurde. Sie wird als eine Spinbox im Objekt Info-Panel angezeigt und kann manipuliert werden.
cyl_unif_d	BoolProperty	Diese Eigenschaft steuert die Anzeige der Spinboxen für das Setzen des Durchmessers eines Zylinders.
cyl_diameter	FloatVectorProperty	Speichert den aktuellen Wert des Durchmessers des Zylinders. Wird im Objekt Info-Panel zur Information neben den Spinboxen für das Setzen des Durchmessers des Zylinders angezeigt.
branch_basis	IntProperty	Anzahl der Verzweigungen die im Internodium anfangen.
branch_succ	BoolVectorProperty	Vektor mit der Information, ob das Internodium zu einer Verzweigung gehört und ob es der erste Knoten in der Verzweigung ist.
branch_name	StringProperty	Diese Eigenschaft speichert den Namen der Verzweigungen, die in dem Internodium anfangen. Sie ist nur dann gesetzt, wenn das Internodium eine Basis für die Verzweigung ist.
axis_name	StringProperty	Diese Eigenschaft speichert den Namen der Achse, zu der das Internodium gehört.
unit_name	StringProperty	Diese Eigenschaft speichert den Namen der Wachstumseinheitengruppe, zu der das Internodium gehört.
plant_name	StringProperty	Diese Eigenschaft speichert den Namen der Pflanzengruppe, zu der das Internodium gehört.

length	FloatProperty	Diese Eigenschaft speichert die Länge des Internodiums. Sie wird nur für die Internodien berechnet, die aus zwei Punkten aufgebaut sind.
n_v_out_lf	IntProperty	Diese Eigenschaft speichert die Anzahl der Vertices die zur Kontur eines Blattes oder einer Blüte gehören. Sie wird zur Sortierung der Dreiecke der konvexen Hülle der Delanay-Triangulation verwendet.
area	FloatProperty	Diese Eigenschaft speichert das Ergebnis der Berechnung der Fläche eines Blattes oder einer Blüte. Sie wird nur für die Internodien berechnet, die aus dem Scan der zusätzlichen Topologieformen stammen.
volume	FloatProperty	Diese Eigenschaft speichert das Ergebnis der Berechnung des Volumens eines Objekts, das aus dem Scan über das Volumen-Panel entstanden ist. Sie kann im Objekt Info-Panel aktualisiert werden.
density	FloatProperty	Diese Eigenschaft speichert den Wert für die Dichte des abgetasteten Objekts. Wenn der Wert gesetzt ist, wird automatisch die Masse des Objekts aus der Dichte und dem Volumen berechnet. Sie kann im Objekt Info-Panel aktualisiert werden.
mass	FloatProperty	Diese Eigenschaft speichert den Wert für die Masse des abgetasteten Objekts. Sie wird aktualisiert nach der Eingabe eines neuen Werts für die Dichte des Objekts.

Das Modul f_ui.py

Das letzte Modul f_ui.py des Addons dient der Darstellung der Komponenten des Addons in der Blender-Oberfläche. Alle hier definierten Klassen erben von bpy.types.Panel, die die Funktionen zum Erstellen eines Panels in Blender anbietet (siehe Seite 30). Die Panels der Erweiterung sind in dem Properties-Fenster im Reiter Scene zu finden. Die poll-Funktionen jeder Panel-Klasse des Addons wurden so implementiert, dass die Panels nur im Objektmodus des 3D-Fensters zu sehen sind. Zu den in f_ui.py definierten Klassen gehören:

- WINDOW_OT_fk_controller_panel
- WINDOW_OT_fk_adjustment_panel
- WINDOW_OT_fk_alignment_panel
- WINDOW_OT_fk_topology_panel
- WINDOW_OT_fk_volume_panel
- WINDOW_OT_fk_o_info_panel
- WINDOW_OT_fk_mtg_panel

Die genaue Beschreibung des Aussehens der Panels, die mit den oben aufgezählten Klassen erstellt werden, ist im Kapitel *Bedienungsanleitung für das Addon* enthalten.

Einführung in Blender

Blender ist eine sehr umfangreiche freie Software mit vielfältigen Einsatzmöglichkeiten. Man kann mit dem Programm 3D-Modelle erstellen, texturieren, rendern und animieren. Entsprechend ist auch die Benutzeroberfläche von Blender sehr komplex. Deshalb werden für das Durchführen der Vermessung aber nur einige wenige Bereiche und Funktionalitäten der Anwendung gebraucht. In diesem Kapitel werden die Grundzüge des Umganges mit Blender geschildert, die in der Vermessung und Bearbeitung des gescannten Modells gebraucht werden. Für weiterführende Informationen werden bei den besprochenen Themen die Verweise zu der Dokumentation oder der Tutorien angeben.

Anpassung der Hauptansicht

Beim ersten Start der Anwendung ist die Standardansicht mit dem Würfel, der Kamera und der Lampe zu sehen. Rechts, links und unter der 3D-Ansicht befinden sich Steuerpanels mit verschiedenen Buttons und Felder, die zur Bedienung der Anwendung gedacht sind (Abb. 28).

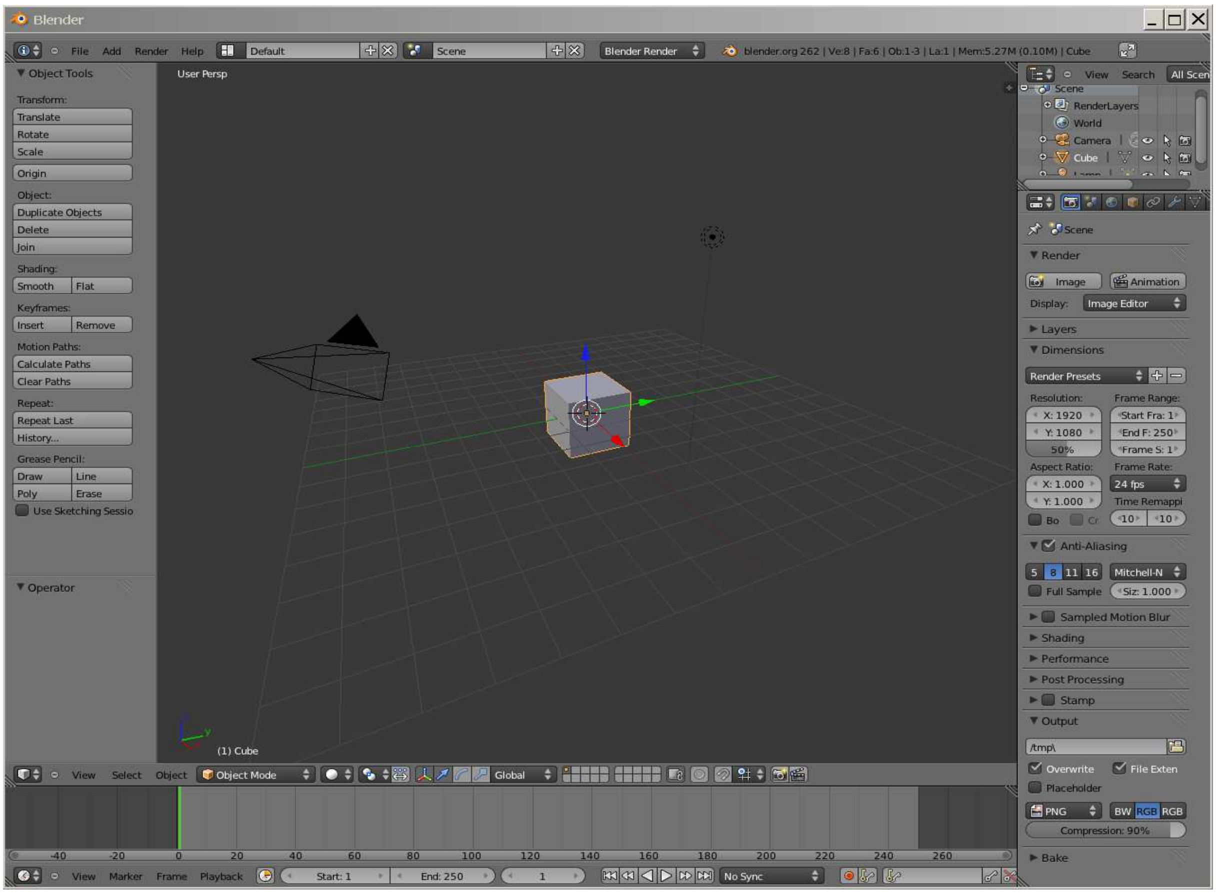


Abb. 28: Standardansicht von Blender. Links befindet sich das Panel für die Bearbeitung des Objekts. In der Mitte ist die 3D-Ansicht des Modells platziert. Rechts-oben sind die Outliner-**Ansicht und direkt darunter der** Preferences-**Bereich des Objektes angesiedelt. In unteren Teil des Fensters befindet sich die** Timeline-**Ansicht zur Steuerung der Animation einer Scene.**

Die meisten Steuerelemente werden bei der Vermessung der Pflanzen nicht gebraucht. Auch nicht alle Standardansichten der Anwendung müssen dabei in dem Hauptfenster zu sehen sein. Blender bietet sehr umfangreiche Möglichkeiten, die Benutzeroberfläche an die eigenen Bedürfnisse anzupassen. Fehlerhafte Anpassungen kann man immer über den Menüpunkt File/Load Factory Settings auf die Standardeinstellungen zurücksetzen. Mit der Menüfunktion File/Save User Settings wird der aktuelle Zustand der Anwendung als Standardeinstellung abgespeichert. Beim nächsten Start wird dieser sofort wieder hergestellt. Die Beispielbilder des Blender-Interfaces, die sich im weiteren Verlauf des Kapitels befinden, haben andere Farbzusammensetzungen als die Standardoberfläche. Dies ist wegen der besseren Sichtbarkeit über File/User Preferences/Themen auf der Presets-Liste einstellbar (Abb. 29).

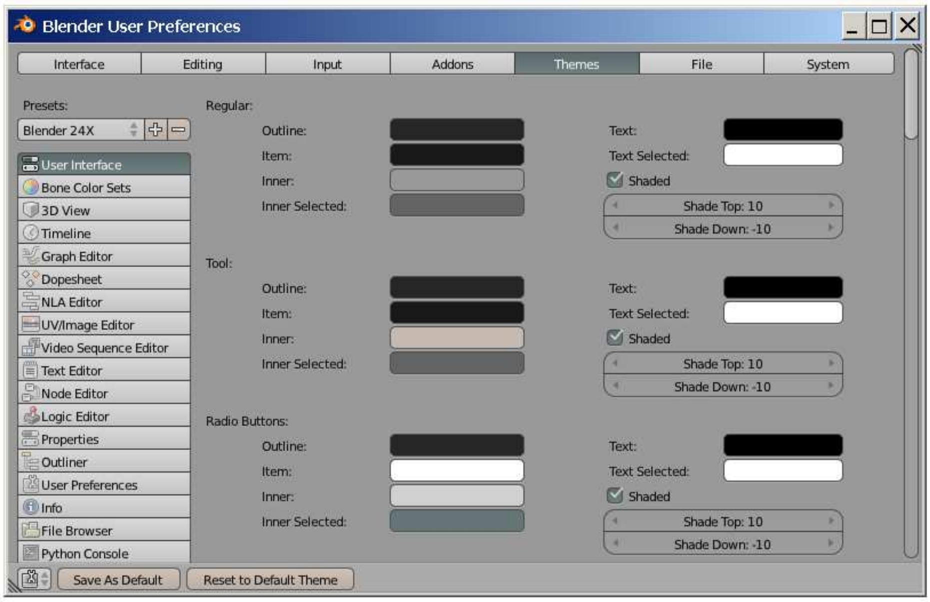


Abb. 29: Fenster für Blender-Einstellungen. Das Fenster kann über den Menüpunkt File/User Preferences **geöffnet werden.**

Für die Visualisierung der vermessenen Daten wird das 3D-Fenster gebraucht. Für die schnelle Übersicht über die erstellten Objekte wird die Outliner-Ansicht (Blender Foundation 2012) benutzt. Hier befindet sich eine Liste aller Objekte, die in der Anwendung angelegt wurden. Die Liste kann man nach verschiedenen Kriterien sortieren. Für die Steuerung und Verwaltung des Scans wird die Properties-Ansicht (Blender Foundation 2012) gebraucht (Abb. 28).

Damit die Einstellungen, die einmal für die Anpassung der Blender-Oberfläche vorgenommen wurden, nicht jedes Mal neu gemacht werden müssen und alle Perspektiven von Blender zur Verfügung bleiben, erstellt man eine neue Perspektive FASTRAK Scan (Abb. 30). Die neue Perspektive muss hinzugefügt werden, bevor man irgendwelche Änderungen vorgenommen hat. Sonst wird die aktuelle Perspektive automatisch auf die aktuellen Einstellungen gesetzt.

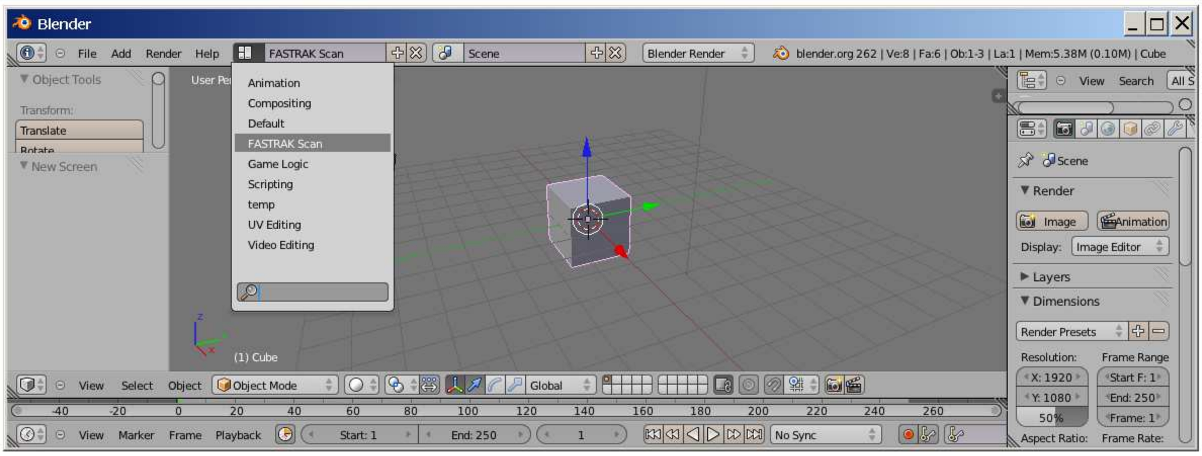


Abb. 30: Erstellen einer neuen Perspektive für die Anpassung der Blender-Oberfläche an das Scannen mit FASTRAK. Die neue Perspektive wird mit dem Pluszeichen, das sich neben der Liste befindet, erstellt und danach FASTRAK Scan **genannt.**

Der erste Schritt der Anpassung ist das Zusammenfügen der 3D-Ansicht und der Ansicht für die Animation des Modells. Durch das Anklicken mit der rechten Maustaste auf der Grenze zwischen den beiden Bereichen wird ein Kontextmenü angezeigt. Mit dem Punkt Join Area werden die Ansichten zusammengefügt. Es bleibt nur die Ansicht, von der der Pfeil weg zeigt. Die andere Ansicht wird ausgeblendet (Abb. 31). Als nächstes wird die Objekt Tools-Ansicht mit der T-Taste ausgeblendet. Danach wird die Outliner-Ansicht in dem linken Bereich der Benutzeroberfläche platziert und die Properties-Ansicht bleibt alleine auf der rechten Seite des 3D-Fensters. Dafür müssen die Outliner-und Properies-Ansicht zuerst zusammengefügt werden. Danach wird die 3D-Ansicht mit dem Split-Befehl des Kontextmenüs, ähnlich wie beim Zusammenfügen, in zwei Teile aufgeteilt. Aus der Liste im neuen Bereich des Fensters wird die Outliner-Ansicht ausgewählt. Als nächstes wird der Scene-Kontext der Properties-Ansicht gewählt ( ) und alle Panels des Kontexts, die für den Scan nicht relevant sind, werden zugeklappt (Abb. 32). Zuletzt wird die 3D-Ansicht angepasst. Der selektierte Standardwürfel wird mit der X-Taste gelöscht. Die Kamera und die Lampe werden auf der Outliner-Liste ( ) ausgeblendet. Damit der Scanbereich immer gut in dem 3D-Fenster sichtbar ist, wird der rechte Bereich des 3D-Fensters, das sogenannte Properties Shelf, mit der N-Taste aufgemacht. Hier werden folgende Parameter eingestellt:

- View-Panel:
  - Clip/Start = 0.01
  - Clip/End = 10000
- Dysplay-Panel:
  - Lines = 40
  - Scale = 10
  - Subdivisions = 10

Danach wird die Ansicht so gedreht, dass die positive Z-Achse nach unten zeigt, was dem Koordinatensystem auf dem Scanner entspricht (Abb. 33). Damit ist die Oberfläche an das Scannen mit FASTRAK angepasst. Man sollte nicht vergessen, die

ganzen Einstellungen mit dem Menüpunkt File/Save User Settings abzuspeichern. Andernfalls muss man die ganze Prozedur beim nächsten Neustart wiederholen.

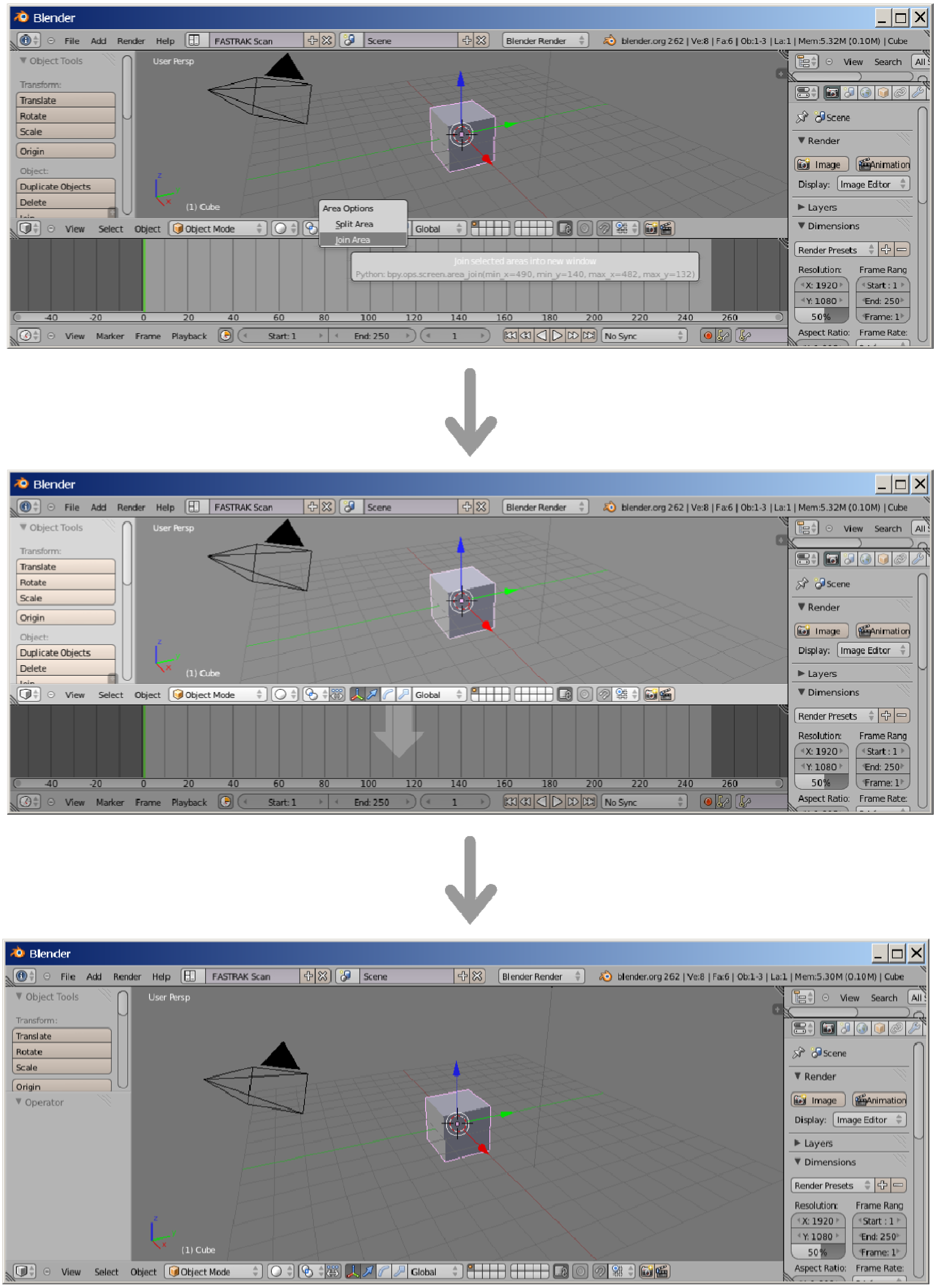


Abb. 31: Zusammenfügen von zwei Blender-Ansichten. Am Ende nimmt die 3D-Ansicht den Platz der Timeline**-Ansicht ein.**

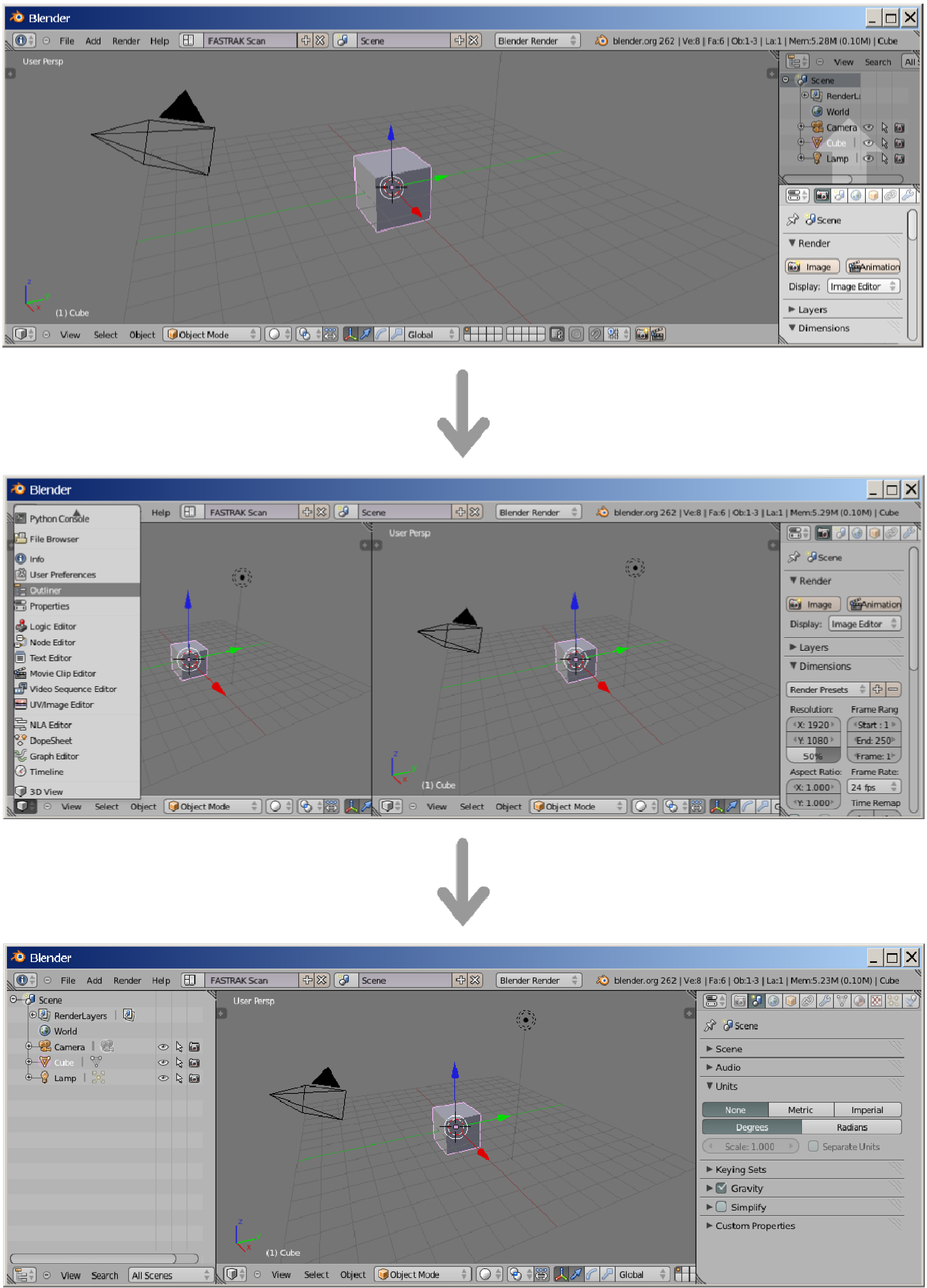


Abb. 32: Aufteilen der Benutzeroberfläche auf die drei für den Scan relevanten Ansichten. Das Setzen des Scene**-Kontexts und Zuklappen von nicht benötigten Panels (drittes Bild).**

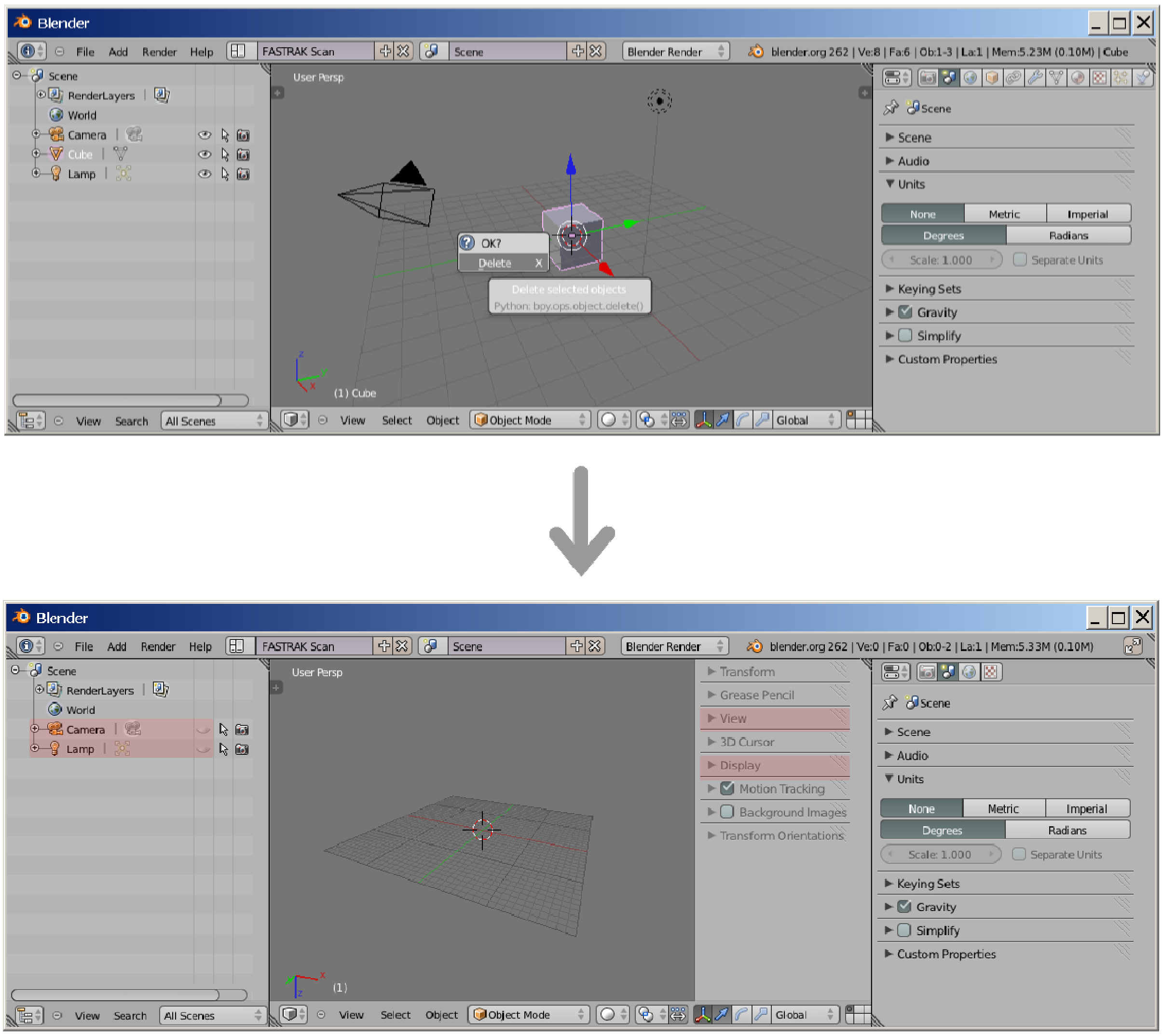


Abb. 33: Anpassung des 3D-Fensters. Die geänderten Bereiche der Oberfläche sind im zweiten Bild rot markiert.

Umgang mit Objekten

Die Objekte in Blender haben drei Grundzustände (Abb. 34). Jedes Objekt kann entweder selektiert oder nicht selektiert sein. Der dritte Zustand eines Objekts heißt: aktiv-Zustand. Während das Programm läuft, ist immer nur ein Objekt in dem aktiv-Zustand versetzt, egal ob es selektiert ist oder nicht. Die meisten Operatoren in Blender, die das Objekt manipulieren können, funktionieren nur bei einem Objekt, das sich in dem aktiv-Zustand befindet. Dies gilt auch für den Scan-Addon.

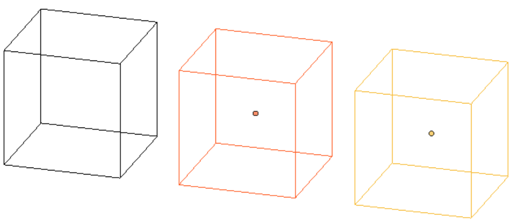


Abb. 34: Drei Zustände eines Objekts in Blender. Links sieht man ein nicht selektiertes Objekt. In der Mitte befindet sich ein selektiertes Objekt und rechts ein selektiertes, aktives Objekt.

Bei der Arbeit an einem Objekt kann man einen von sechs angebotenen Editiermodi wählen. Der einzige Modus mit dem das Abtasten der Objekte möglich ist, ist der Object Mode. Bei der Vermessung wird auch von Zeit zu Zeit der Edit Mode gebraucht. In diesem Modus kann das Gitter des Objekts bearbeitet werden. Das Umschalten zwischen den beiden Modi kann über die Oberfläche oder mit der Tabulator-Taste gemacht werden (Abb. 35).

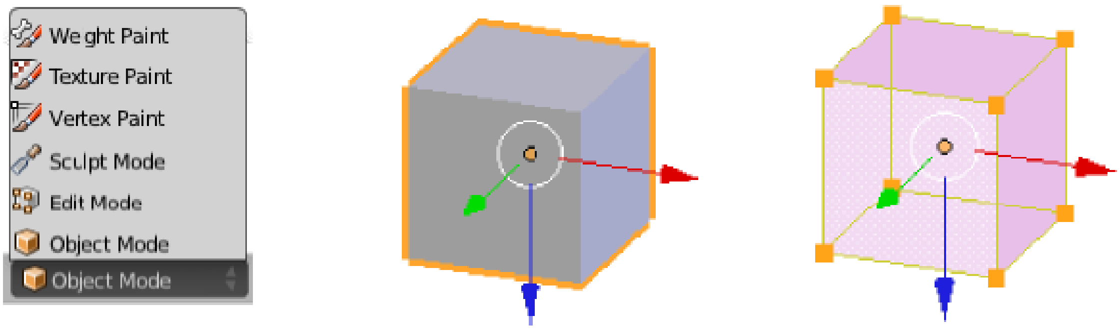


Abb. 35: Sechs verschiedene Bearbeitungsmodi eines Gitterobjekts. Die Liste befindet sich, mit anderen Werkzeugen, im unteren Bereich der 3D-Ansicht. Der linke Würfel entspricht der Standardansicht eines selektierten Objekts im Objekt Mode**. Der Würfel rechts ist im** Edit Mode **zu sehen.**

Die Objekte können in vier verschiedenen Modi angezeigt werden. Am häufigsten wird bei der Messung der Solid- und Wireframe-Modus benutzt. Mit der Z-Taste kann man schnell zwischen den beiden Modi umschalten. Alternativ kann dies auch über die Werkzeuge der 3D-Ansicht gemacht werden (Abb. 36).

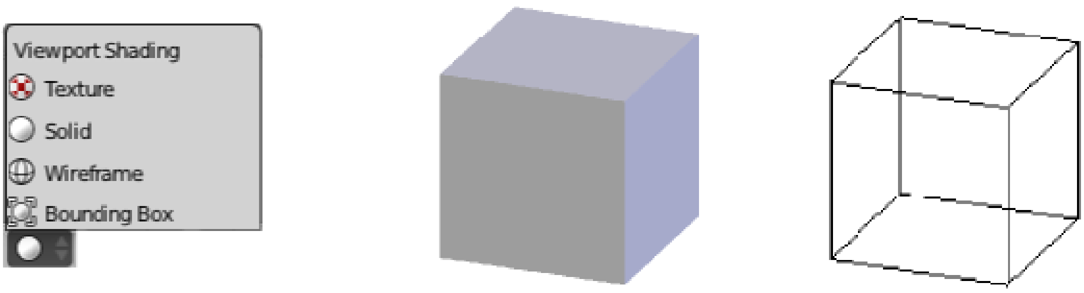


Abb. 36: Vier verschiedene Anzeigemodi des Objekts. Liste befindet sich, mit anderen Werkzeugen, im unteren Bereich der 3D-Ansicht. Der ausgefüllte Würfel entspricht der Anzeige im Solid**-Modus. Der Würfel rechts entspricht der Anzeige im** Wireframe**-Modus.**

Das Selektieren der Objekte und der Teile des Objektsgitters kann man sowohl im Objekt Mode als auch im Edit Mode mit der rechten Maustaste mit Ctrl und der linken Maustaste, per Tastatur mit A oder B erfolgen. Die gute Beschreibung der Methoden für Selektieren und Deselektieren der Objekte in Blender befindet sich im Blender-Wiki (Blender Foundation 2012).

Für die Vermessung und Bearbeitung gescannter Objekte ist es wichtig zu wissen, wie man das Gitter des Objekts editieren kann und wie die Richtungen der Normalenvektoren der Polygone des Gitters eingestellt werden können. Dies ist

nur für das aktive Objekt im Edit Mode möglich. Beim Editieren lässt sich einstellen, ob die Vertices, Kanten oder Polygone zur Bearbeitung freigestellt werden sollen ( ). In diesem Modus kann man eventuelle Lücken im Gitter ausfüllen sowie falsch aufgenommene Gitterpunkte oder falsch gezeichnete Polygone korrigieren. Nachdem man z. B. drei Vertices der Lücke im Gitter selektiert hat, kann man sie mit der Taste F zu einem Polygon verbinden lassen (Abb. 37). Zum Entfernen von Vertices, Kanten oder Polygone muss man zuerst die entsprechenden Teile des Gitters im Edit Mode selektieren und danach die Taste X drücken. Dadurch wird ein Kontextmenü angezeigt. Aus dem Menü muss man einen passenden Befehl zum Löschen der Elemente des Gitters wählen (Abb. 37). Alle Aktionen, die beim Editieren des Gitters unternommen werden können, lassen sich auch über die Werkzeugliste im unteren Bereich der 3D-Ansicht ausführen.

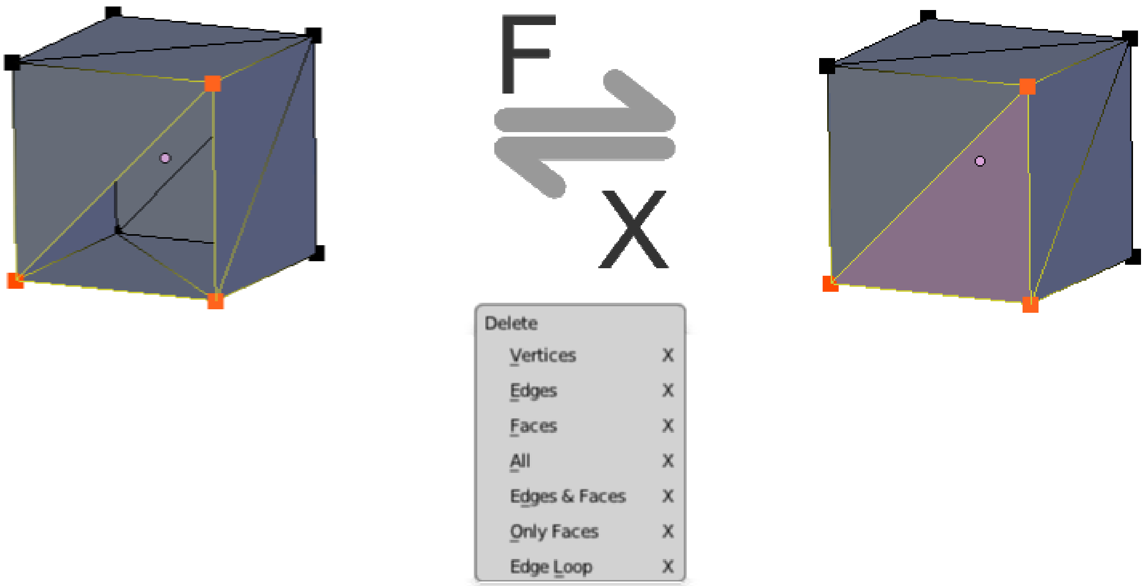


Abb. 37: Hinzufügen und Löschen eines dreieckige Polygons des Objektgitters. Nachdem die Vertices selektiert wurden, wird es mit der Taste F **oder über den Werkzeugmenüpunkt** Mesh/Faces/Fill **zum Gitter hinzugefügt. Das Löschen der Gitterelemente kann mit der Taste X oder über den Werkzeugmenüpunkt** Mesh/Delete . . . **gemacht werden.**

Die Normalenvektoren kann man im Edit Mode über die rechte Werkzeugliste der 3D-Ansicht anzeigen. Diese Liste wird mit der Taste N oder über das Pluszeichen rechts oben in dem Fenster eingeblendet. Die Einstellungen für die Anzeige der Normalenvektoren befinden sich in dem Panel Mesh Display unter dem Punkt Normals/Faces und Normals/Normal size. Die Operatoren für das Manipulieren der Normalenvektoren sind an der linken Seite des 3D-Ansicht in dem aufklappbaren Panel unter dem Punkt Normals untergebracht. Hier kann man die Normalen neu berechnen lassen oder deren Richtung ändern. Alternativ kann man die Einstellungen für die Normalenvektoren und Erstellung der Polygone mit Hilfe der Werkzeugliste der 3D-Ansicht über den Menüpunkt Mesh/Faces wählen (Abb. 38).

Verwaltung der Erweiterungen

Bevor die Vermessung mit dem FASTRAK-Scanner über die Blender-Oberfläche möglich ist, muss die Erweiterung in Blender zur Verfügung gestellt werden. Dafür muss der Quelltext des Addons im richtigen Zielverzeichnis der Anwendung liegen (siehe Seite 34 und 48). Wenn alles richtig vorbereitet wurde, kann man in dem Blender User *Preferences*-Fenster in den Reiter Addons wechseln und die Erweiterung ankreuzen (Abb. 39). Man findet sie unter der Kategorie *3D-View* unter den Namen *3D Plant Scan with FASTRAK*.

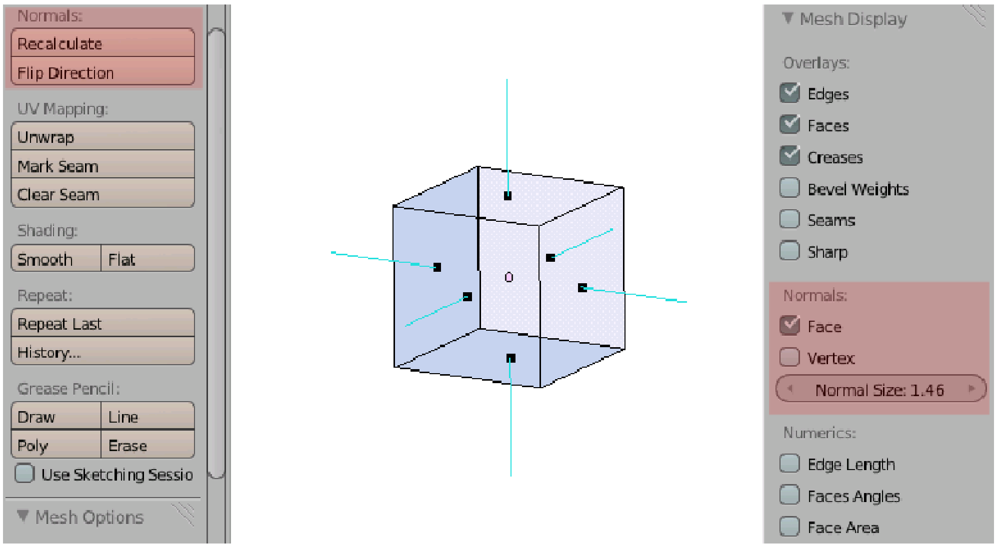


Abb. 38: Anzeige und Manipulation der Normalenvektoren der Polygone. Die Menüpunkte, mit denen man die Normalenvektoren anzeigen und editieren kann, sind rot hervorgehoben. Die Einstellungen sind nur im *Edit Mode* **verfügbar.**

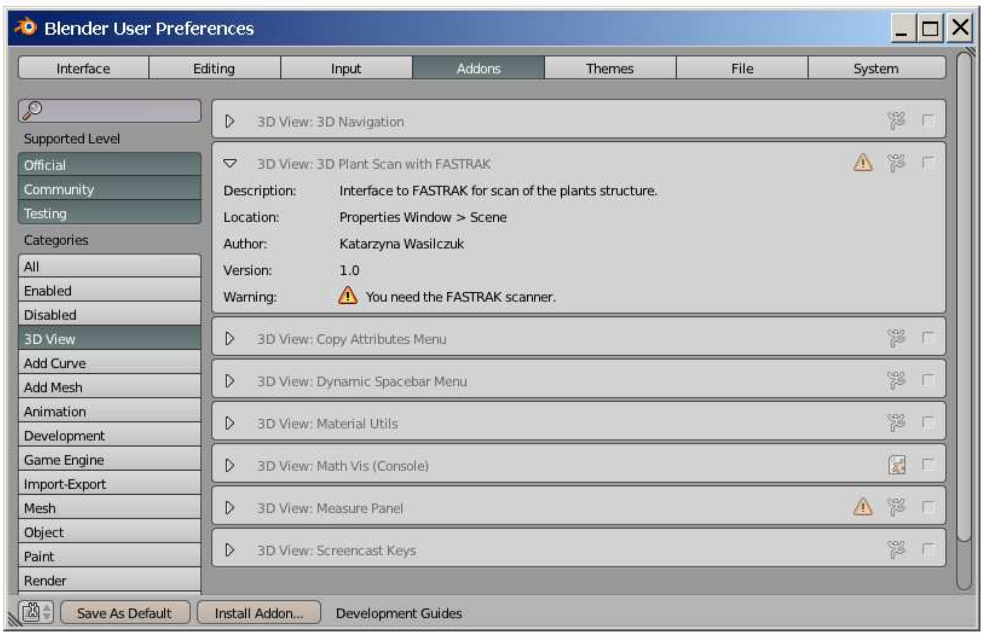


Abb. 39: Fenster zur Verwaltung der Addons. Die Erweiterung wird nach dem Ankreuzen der Checkbox an Blender angebunden. Beim Starten ist es wichtig, dass der Scanner mit dem Rechner verbunden und eingeschaltet ist. Da die Verbindung an dem Port exklusiv nur für eine Instanz reserviert ist, darf keine andere Anwendung mit dem Scanner verbunden sein. Ohne Verbindung zum Scanner werden nur die Teile des Addons angezeigt, die Offline verfügbar sind.

Bedienungsanleitung für das Addon

In diesem Kapitel wird die Benutzung der zur Verfügung gestellten Funktionalitäten erklärt. Die Erweiterung benutzt drei Bereiche der Blender-Oberfläche: das Outliner-Fenster, die 3D-Ansicht und das Properties-Fenster. Das Addon selbst besteht aus sieben verschiedenen Panels, die alle sich im Properties-Fenster befinden (Abb. 40).

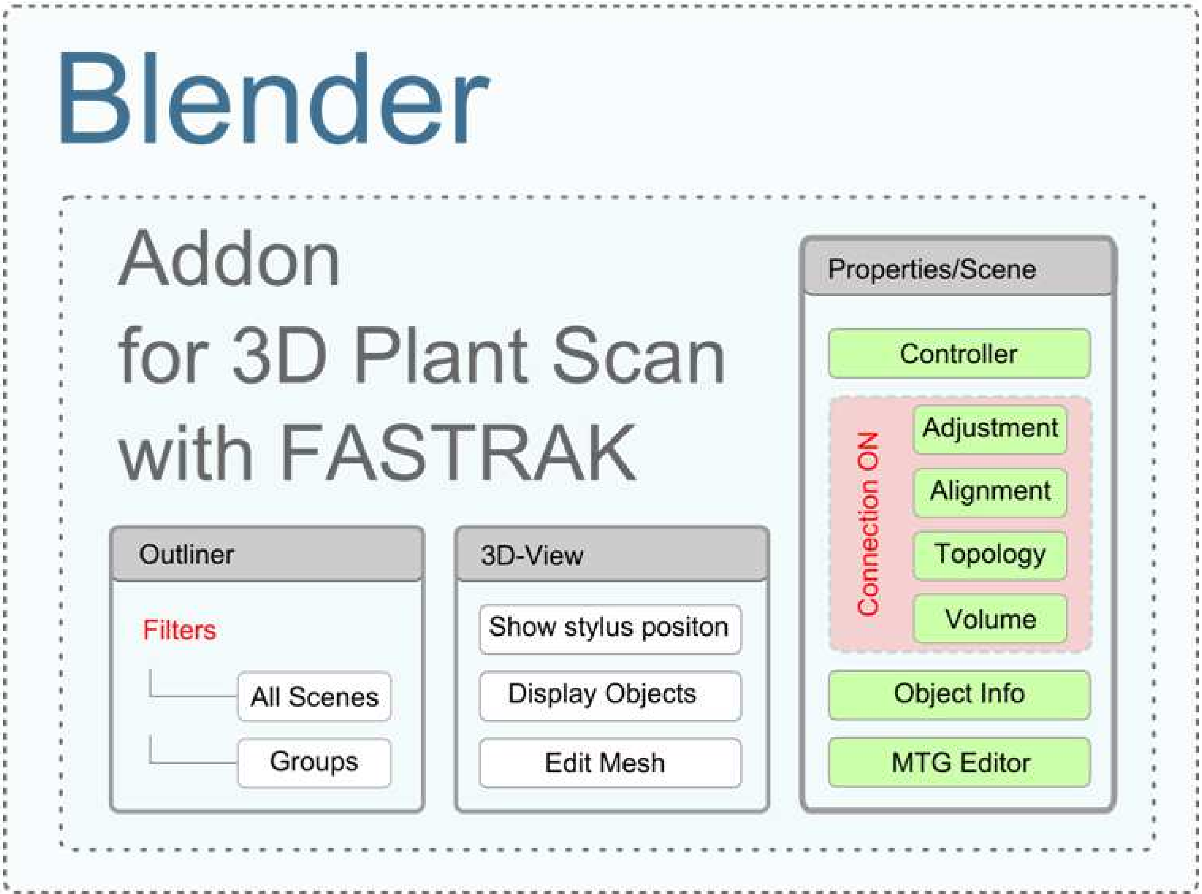


Abb. 40: Platzierung des Addons in Blender und Elemente der Blender-Oberfläche und dessen Funktionalitäten, die zur Darstellung der gescannten Objekten benutzt wurden.

Im Outliner befindet sich die Liste mit allen in Blender angelegten Objekten. Die Liste kann nach verschiedenen Kriterien sortiert werden, wobei für die Vermessung nur die Filter All Scenes und Groups interessant sind. In der 3D-Ansicht werden die gescannten Objekte und die Position des Stifts im Raum angezeigt. Falls nötig, kann man hier das erstellte Gitter des Objekts editieren. Im Properties-Fenster im Reiter Scene befinden sich alle Panels der Scanerweiterung. Das Addon selbst besteht aus sieben verschiedenen Panels. Dabei dienen vier der Panels dem Sammeln der Datenpunkte und der Steuerung der aktuellen Art der Messung (Adjustment, Alignment, Topology, Volume). Die Panels werden angezeigt, wenn die Verbindung zum Scanner geöffnet ist. Dabei ist jeweils nur eines der vier

Panels sichtbar. Dies kann über eine Auswahlliste im Panel FASTRAK: Controller gesteuert werden. Das Panel ermöglicht außerdem die Auswahl der Parameter für den gewählten Vermessungstyp, das Löschen der ausgewählten Objekte und Ein- und Ausblenden der Objektnamen sowie das Exportieren der Scandaten in CSV- oder MTG-Format (Abb. 44, Abb. 45). Zwei weitere Panels dienen der Anzeige sowie der Verwaltung und Bearbeitung der Metainformationen zu den gescannten Objekten (Objekt Info, MTG Editor). Sie werden angezeigt, sobald die Objekte aus dem Scan in der Anwendung angelegt wurden und hängen nicht von der Verbindung mit dem Scanner ab.

Kontrollpanel

Wenn die Erweiterung im User Preferences-Fenster ausgewählt wurde, wird im Standardfall nur das Panel FASTRAK: Controller in der Blender-Oberfläche angezeigt. Dabei ist zu beachten, dass alle Panels der Erweiterung nur im Objekt Mode eingeblendet werden. Das Kontrollpanel befindet sich im Reiter Scene des Properties-Fensters und ist in drei Bereiche aufgeteilt (Abb. 41).

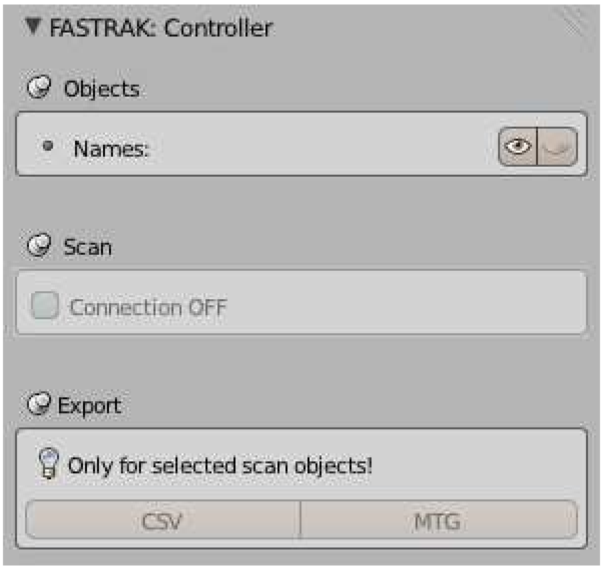


Abb. 41: Basisansicht des Kontrollpanels der Erweiterung für das Scannen von Pflanzen. Die Verbindung zum Scanner ist geschlossen. Der Scanner ist jedoch an der seriellen Schnittstelle zur Zeit des Addon-Starts verfügbar.

Im ersten Bereich (Objects) kann man die Namen der in der 3D-Ansicht selektieren Objekte ein- und ausblenden ( ). Bei der Benutzung des Addons ist es wichtig, dass die Scanobjekte nicht direkt mit den Blender-Operatoren gelöscht werden, sondern mit denen aus der Erweiterung. Der Grund dafür ist, dass beim Löschen der Objekte mit den Blender-Operatoren keine Informationen für die Verwaltung des Scans aktualisiert werden.

Der zweite Bereich dient der Verwaltung der Vermessung und der Verbindung mit dem Scanner. Falls beim Registrieren des Addons die Verbindung mit dem Scanner nicht erfolgreich war, wird eine Information darüber hier eingeblendet und eine Möglichkeit zum erneuten Verbindungsversuch ( ) gegeben (Abb. 42).



Abb. 42: Zustand der Benutzeroberfläche des Kontrollpanels, wenn der Scanner während des Startens der Erweiterung nicht verfügbar war.

Wenn der Scanner verfügbar ist, kann man die Verbindung öffnen und die Vermessung anfangen. Je nach Typ der Vermessung, der aus der Liste Scan Type ausgewählt werden kann, werden entweder zwei oder ein zusätzlicher Parameter für das Senden und Darstellen der gescannten Punkte eingeblendet. Der erste Parameter Cursor update (Abb. 43) dient zum Setzen der Häufigkeit mit der während des Scans die Position des 3D-Cursors (in der 3D-Ansicht aktualisiert werden soll. Je kleiner der Wert ist, desto öfters wird ein Update der Blender-Oberfläche erfolgen. Wenn sehr viele Daten über die serielle Schnittstelle gesendet werden, kann das häufige Aktualisieren der Position des Cursors zur Verlangsamung des Zeichnens der Objekte führen. Der zweite Parameter ist nur bei den Vermessungstypen zu sehen, bei denen das Abtasten der Pflanzen im kontinuierlichen Modus des Scannens stattfindet. Der Parameter Distance (Abb. 43) dient zur Einstellung des Abstandes zwischen zwei nacheinander vom Scanner geschickten Punkten. Wenn z.B. der Abstand von 1 cm gewählt wurde, sendet der Scanner einen Punkt nur dann, wenn sich die Spitze des Empfängerstifts in eine Richtung mindestens um 1cm bewegt hat.

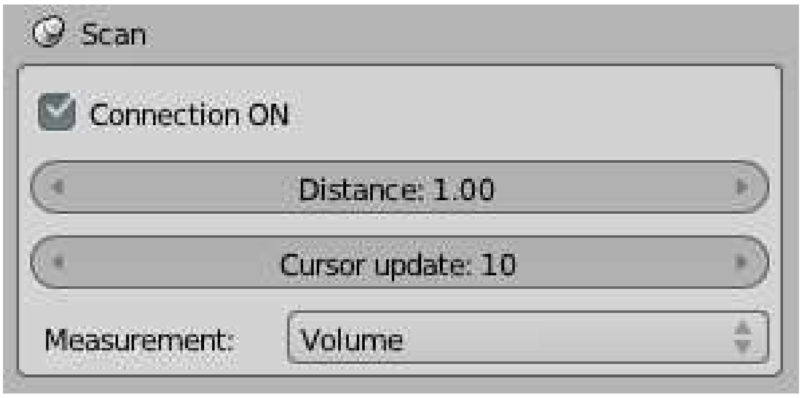


Abb. 43: Ansicht des Kontrollpanels bei einer geöffneten Verbindung. Der Parameter Cursor update ist immer eingeblendet. Der Parameter Distance wird nur für Messungen im kontinuierlichen Modus angezeigt. In der Measurement-Liste sind vier Möglichkeiten zu wählen: Adjustment, Alignment, Topology, Volume.

Der dritte Bereich des Kontrollpanels ermöglicht das Exportieren der Daten der gescannten Objekte (Abb. 41). Die Export-Operatoren kann man auch über das Menü File/Export aufrufen. Zum Exportieren der Daten wird ein neues Fenster von Blender angezeigt (Abb. 44). Hier muss man den Pfad und den Namen für die Exportdatei angeben. Es werden nur die Informationen zu den selektierten Objekten, die aus dem Scan stammen, weggeschrieben. Die Export-Operatoren sind nur bei ausgeschalteter Verbindung im Panel eingeblendet.

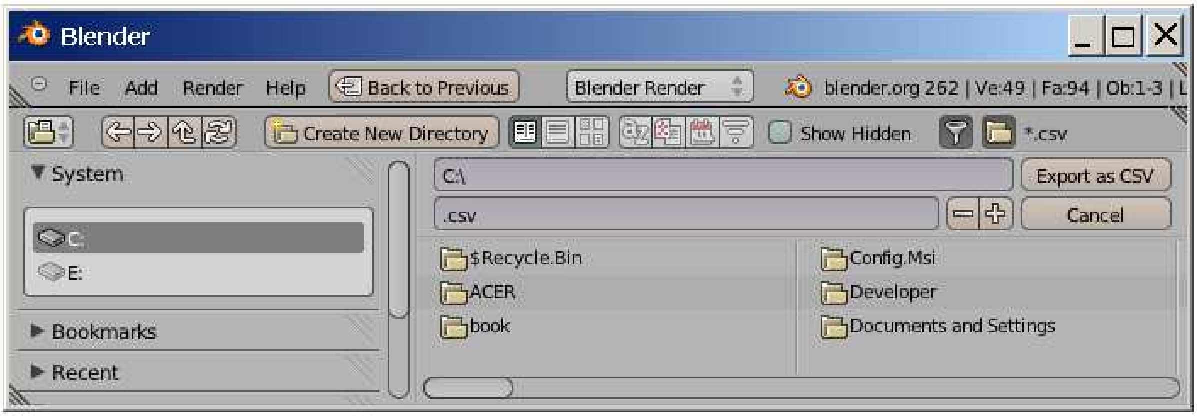


Abb. 44: Blender-Exportfenster. Ansicht für den CSV-Export.

Beim CSV-Export werden alle Informationen zu den selektierten Objekten weggeschrieben (Abb. 45). Der MTG-Export ist zurzeit in der Erweiterung nur auf dem Level der Internodien möglich. Dabei werden nur Informationen über dem Namen, der Länge und Durchmesser sowie über dem Typ abgespeichert.

Blender - CSV export						
Date	Sun, 28 Oct 2012 18:45:35					
Name	A1/N1					
Type	Internode					
Scan date	28.10.2012 at 18:44:20					
Comment						
Topology (axis, unit, plant)	A1	null	null			
Basis for n branches	0					
First successor in branch	False					
Diameter	10					
Point.1 (x, y, z, yaw, pich, roll)	8,329999924	16,35000038	-5,340000153	-143,2100067	28,65999985	162,2899933

Abb. 45: Beispieldatei für ein CSV-Export, die in Excel importiert wurde.

Scan der Struktur einer Pflanze

Für die Vermessung der Topologie von Pflanzen werden zwei alternative Scanmethoden angeboten. Die Kontrollflächen werden angezeigt, sobald eine neue Achse erstellt wurde (Abb. 46). Die normale Vermessung wird mit Hilfe des Reiters Standard im Einzelpunktmodus des Scanners durchgeführt. Einzelpunktmodus bedeutet, dass erst nach der Betätigung der Stifttaste oder nach der Aufforderung aus dem Programm ( ) ein Punkt vom Scanner aufgenommen und in der 3D-Ansicht dargestellt wird. Die andere Möglichkeit der Vermessung von Pflanzenteilen wie Blätter, Blüten oder Früchten ist im Reiter Additional zur Verfügung gestellt. Hier wird der kontinuierliche Modus des Scanners eingesetzt. In diesem Modus werden vom Scanner die Punkte nach einem bestimmten Abstand, der im Kontrollpanel einstellbar ist (siehe Seite 116 und Abb. 43), gesendet. Sowohl im ersten als auch im zweiten Reiter des Topology-Panels ist eine Information zum aktuellen Stand der Vermessung. In der Liste State befindet sich am ersten Platzt die Ordnungsnummer der Achse, danach folgt die Anzahl der Internodien auf der Achse und am letzten Platz steht die Information über die gesamte Anzahl der

abgetasteten und dargestellten Internodien. Beispielweise [2, 3, 20] würde dann bedeuten, dass die letzte hinzugefügte Achse die Nummer 2 hat, dabei sind schon 3 Internodien auf der Achse vorhanden und die Anzahl der im ganzen Modell aufgenommen Internodien 20 beträgt. Eine neue Achse kann man mit dem Pluszeichen hinzufügen. Mit dem Minuszeichen wird sie zusammen mit den zu ihr gehörenden Internodien entfernt ( ). Das zuletzt gezeichnete Internodium kann mit dem Operator Delete last node des Bereichs Node control sowohl im Standard-Reiter als auch im Additional-Reiter aus der Achse gelöscht werden.

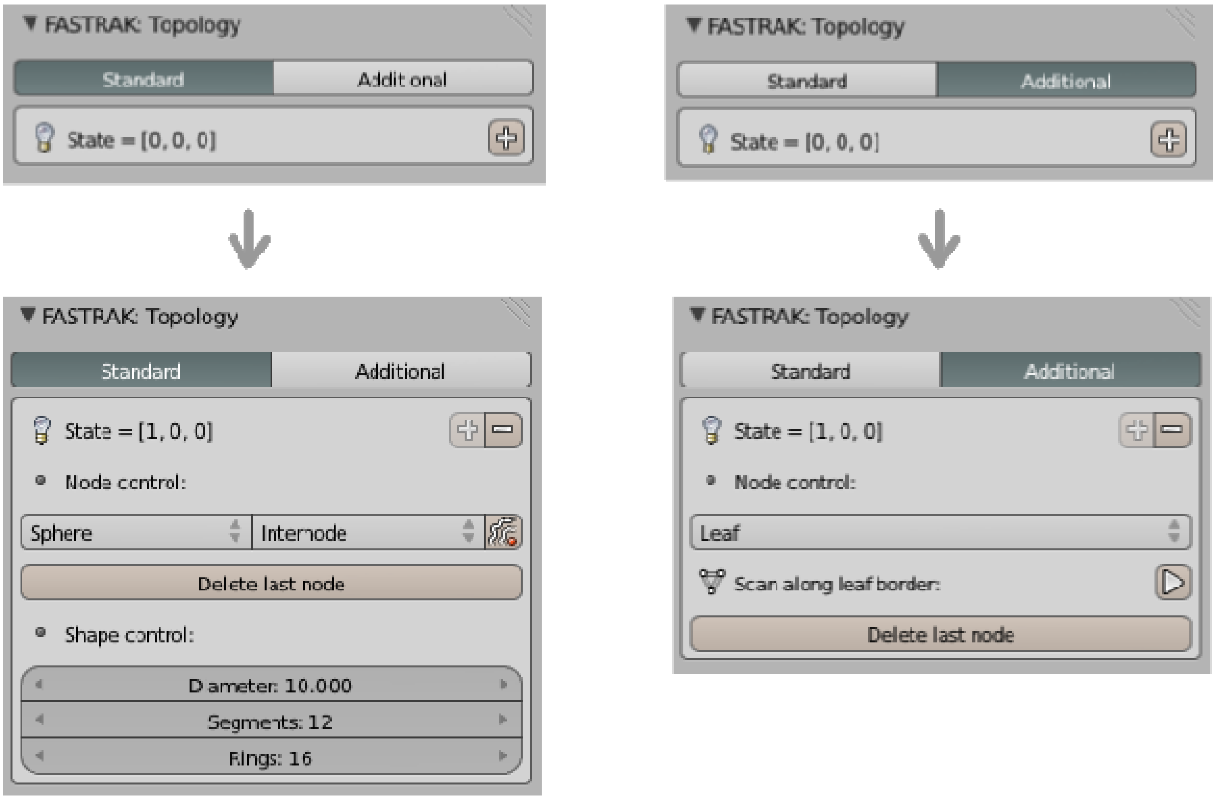


Abb. 46: Zwei Alternativen für die Vermessung der Struktur von Pflanzen. In dem Reiter Standard **(rechts) wird der Scanner im Einzelpunktmodus eingesetzt. Der Reiter** Additional **(links) dient zur Vermessung der Blätter, Blüten oder Früchte im kontinuierlichen Modus des Scanners.**

Zeichnen der standardmäßigen Topologie

Im Bereich Node control kann man die abgemessenen Punkte als eine von den drei Geometrieformen, Sphäre, Linie oder Zylinder in der 3D-Ansicht visualisieren. Dabei kann man den Typ des zu messenden Internodiums aus der Node Type-Liste, die sich an der Liste der Geometrieformen befindet, wählen. Es werden hier vier Möglichkeiten angeboten: Internode, Leaf, Flower und Fruit. Je nach Wahl ist der Name des neuen Objekts unterschiedlich zusammengesetzt (Tab. 7).

Der Bereich Shape control dient der Einstellung der Parameter für das Zeichnen der geometrischen Formen. Der wichtigste Parameter ist der Durchmesser für die zu zeichnende Geometrie. Das Durchmessereingabefeld ist nur für die Sphäre und den Zylinder eingeblendet. Der nächste Parameter Segments ist auch für beide Formen einstellbar. Mit diesem Parameter kann man die Anzahl der vertikalen Segmente der zu zeichnenden Geometrie bestimmen. Das letzte Eingabefeld Rings

wird nur für eine Sphäre eingeblendet und erlaubt die Anzahl der horizontalen Segmente der Kugel festzulegen. Der zu zeichnende Zylinder oder die Sphäre werden um die abgemessenen Punkte erstellt. Im Falle einer Sphäre werden die Koordinaten des Messpunkts dem Schwerpunkt der Geometrie zugeordnet. Im Falle eines Zylinders wird der erste Punkt als die Mitte des unteren Deckels und der zweite Punkt als die Mitte des oberen Deckels definiert (Abb. 47). Wenn ein Zylinder als erstes Internodium auf der Achse gezeichnet werden soll, müssen zwei Punkte abgemessen werden, sonst werden die Zylinder mit nur einem neuen Punkt gezeichnet. Koordinaten des Punktes, der davor gemessen wurde, werden zum Zeichen des oberen Deckels des Zylinders verwendet. Das Prinzip wird auch für das Zeichnen der Linie benutzt.

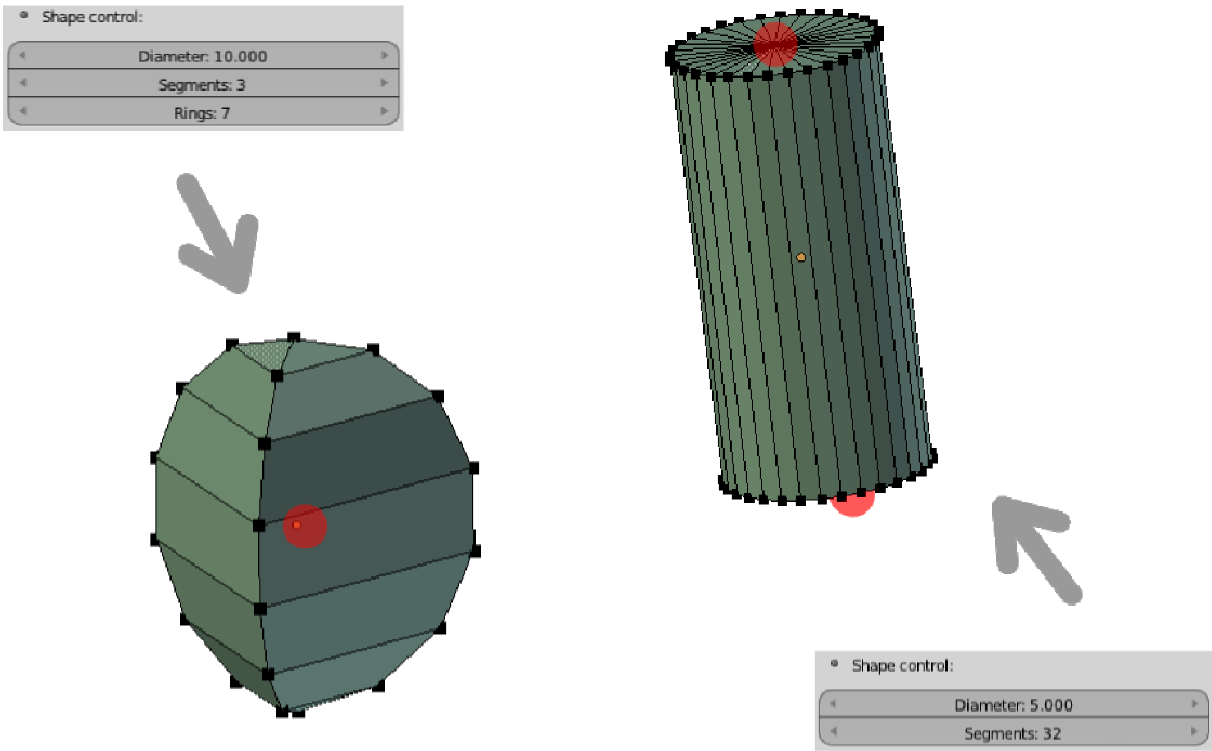


Abb. 47: Beispiel für die Einstellung der Parametern für Anzahl der Segmente und der Ringe. Links befindet sich eine Sphäre und rechts ein Zylinder. Die echten Messpunkte sind rot hervorgehoben.

Zeichnen der zusätzlichen Formen für Blätter, Blüten und Früchte

Die Erstellung der geometrischen Form für Blätter, Blüten und Früchte im Reiter Additional beruht auf einem anderen Prinzip. Die Punkte werden im kontinuierlichen Modus des Scanners gesammelt. Jede Aktion die der Benutzer mit dem Stift steuern muss, wird mit einem Tonsignal bestätigt (Abb. 48). Der erste Schritt bei einer Vermessung eines Blattes oder einer Blüte ist das Abtasten der äußeren Kontur des Objekts. Die Aufnahme der Kontur kann beliebig oft mit dem Operator für das Anhalten des Messung ( ) gestoppt werden und mit dem Startoperator ( ) fortgesetzt werden.

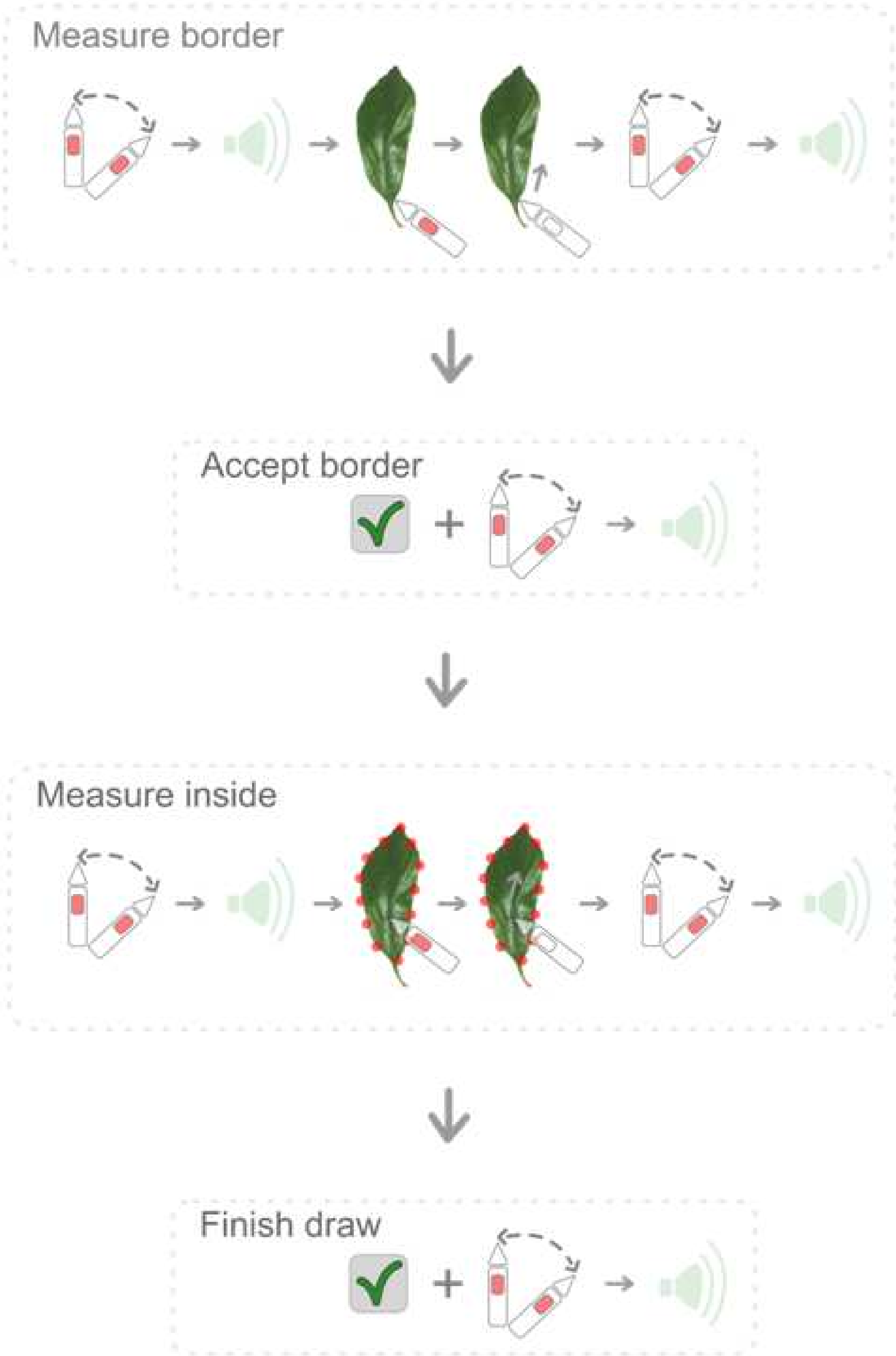


Abb. 48: Vorgehensweise zum Vermessen eines Blattes. Die gedrückte Stifttaste ist rot markiert. Je nachdem, wie groß der Parameter Distance **(siehe Seite 116) eingestellt wurde, wird für das Start- und Stoppsignal sowie die Bestätigung ein verschiedenes Ausmaß für die Stiftbewegung gebraucht, bis der Scanner ein Punkt sendet und die nächste Aktion beginnen kann.**

Der Rand muss immer gegen den Urzeigersinn abgemessen werden. Wenn man innerhalb der Kontur weitere Punkte im Uhrzeigersinn abmisst, wird bei der Vernetzung ein Loch im Gitter entstehen. Falls während der Messung einige von den Messpunkten gelöscht werden sollen, kann man in den Edit Mode des Objekts wechseln und die Korrekturen durchführen (Abb. 37). Wenn die Messung des Randes abgeschlossen wurde, muss sie bestätigt werden (☑). Nach der Bestätigung dürfen keine weiteren Punkte aus dem Rand gelöscht werden. Der zweite Schritt ist das Vermessen des Inneren des Blattes oder der Blüte. Hier wird mindestens ein Punkt gebraucht. Die Messung kann, wie bei der Konturaufnahme, jede Zeit angehalten und an anderer Stelle weiter geführt werden. Wenn die Aufnahme der Punkte abgeschlossen wurde, muss man dies bestätigen. Nach der Bestätigung startet die Berechnung der Verbindungen zwischen den Messpunkten und anschließend das Ausfüllen des Gitters (siehe Seite 84). Das Steuern der Aktionen für das Starten und die Pause ist mit den Bewegungen des Stiftes gekoppelt, damit die Person, die an der Pflanze steht, nicht ständig zum Rechner laufen muss. Das Akzeptieren der Messung ist teilweise mit der Bewegung des Stiftes und teilweise über die Benutzeroberfläche gesteuert (siehe Seite 73 und Abb. 48). Die letzte Besonderheit dieser Vermessungsart ist, dass zurzeit für die Vernetzung der abgemessenen Punkte bei Blättern und Blüten die Punktwolke auf eine 2D-Ebene projiziert wird. Deshalb soll man bei der Aufnahme darauf achten, dass die abgetasteten Punkte sich nicht überlappen.

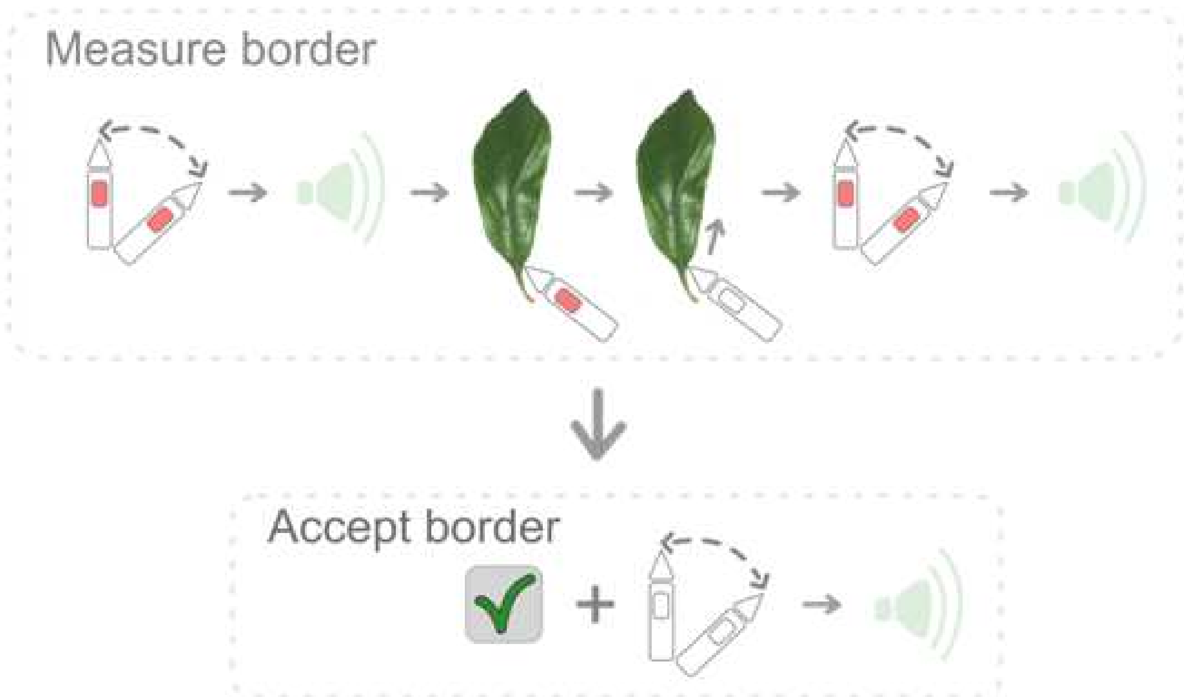


Abb. 49: Vorgehensweise bei Vermessung einer Frucht. Die gedrückte Stifttaste ist rot, und die losgelassene weiß, dargestellt. Je nachdem wie groß der Parameter Distance **(siehe Seite 116) eingestellt wurde, wird für den Start- und Stoppsignal sowie die Bestätigung ein verschiedenes Ausmaß für die Stiftbewegung gebraucht bis der Scanner ein Punkt sendet und die nächste Aktion beginnen kann.**

Die Vermessung einer Frucht im Additional-Reiter verläuft ähnlich wie die Vermessung eines Blattes oder einer Frucht. Es wird aber nur die Oberfläche der Frucht abgetastet. Die Messung kann unterbrochen werden. Vor der Bestätigung darf man die angezeichneten Punkte löschen (Abb. 37). Wenn die Aufnahme abgeschlossen und bestätigt wurde (+ Bewegung des Stiftes ohne Drücken der Taste), wird der Algorithmus zur Vernetzung der Punktwolke gestartet. Je nach Anzahl der aufgenommenen Punkte kann dies länger oder kürzer dauern. Die Vorgehensweise bei der Vermessung einer Frucht ist in (Abb. 49) dargestellt.

Korrektur der Daten

Die Erweiterung erlaubt es einem Benutzer zwei Arten von Korrekturen vorzunehmen. Zum einem kann er die Verzerrung des magnetischen Feldes korrigieren und zum anderen die Entzerrung der Koordinaten nach der Verschiebung des Senders einsetzen. Die die für die Berechnung der entsprechenden Koeffizienten nötigen Punkte werden über das Panel FASTRAK: Adjustment aufgenommen.

Kalibrierung

Die Kalibrierung kann erst eingesetzt werden, wenn die Koeffizienten des Korrekturpolynoms für die x-, y- und z-Koordinate ermittelt sind. Dafür muss die Referenzbox mit Hilfe des FASTRAK: Adjustment Panels vermessen werden (Abb. 50). Die Messvorrichtung und deren standardmäßige Platzierung im Bezug zum Sender ist in (Abb. 52) dargestellt und beschrieben. Wenn die Hinweise zu der Platzierung eingehalten werden, kann die erste Ansicht ohne Änderungen akzeptiert werden. Falls die Kalibrierungsbox anders aufgestellt wurde, muss man den Verschiebungsvektor vom Ursprung des Senderkoordinatensystems entsprechend setzen. Der Ursprung befindet sich im Mittelpunkt des Senders. Wenn der Verschiebungsvektor akzeptiert wurde, kann man nacheinander die Punkte der Referenzbox von links nach rechts, und vom Sender weg, messen. Auf den Teilen der Kalibrierungsvorrichtung sind entsprechende Messpunkte farbig markiert. Nachdem alle Punkte (3 Ebenen zu je 16 Punkten) eingescannt wurden, werden nach der Bestätigung ( ) die Koeffizienten ermittelt (siehe Seite 95) und das Gitter gezeichnet. Danach kann im Panel FASTRAK: Controller die Online Kalibrierung der Punkte ein- und ausgeschaltet werden (Abb. 51).

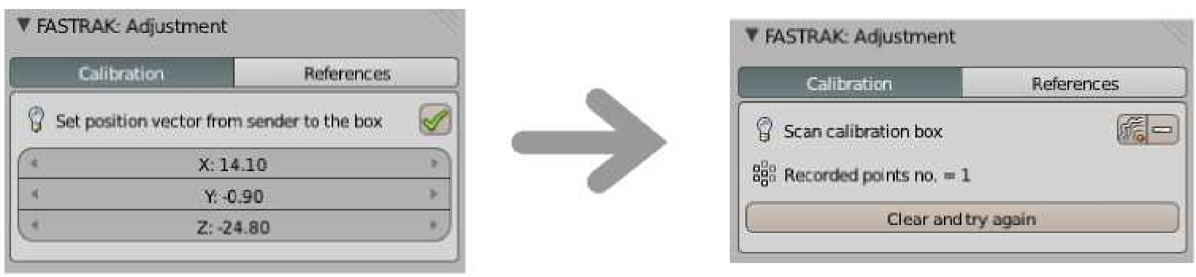


Abb. 50: Reiter für die Aufnahme der Referenzbox für die Kalibration.

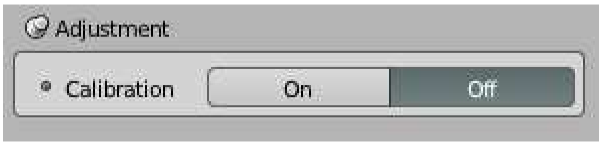


Abb. 51: Nach der Berechnung der Koeffizienten für die Kalibrierung wird der Adjustment-Steuerungsbereich in dem Kontrollpanel eingeblendet. Standardmäßig ist die Online-Korrektur der neu ankommenden Daten ausgeschaltet.

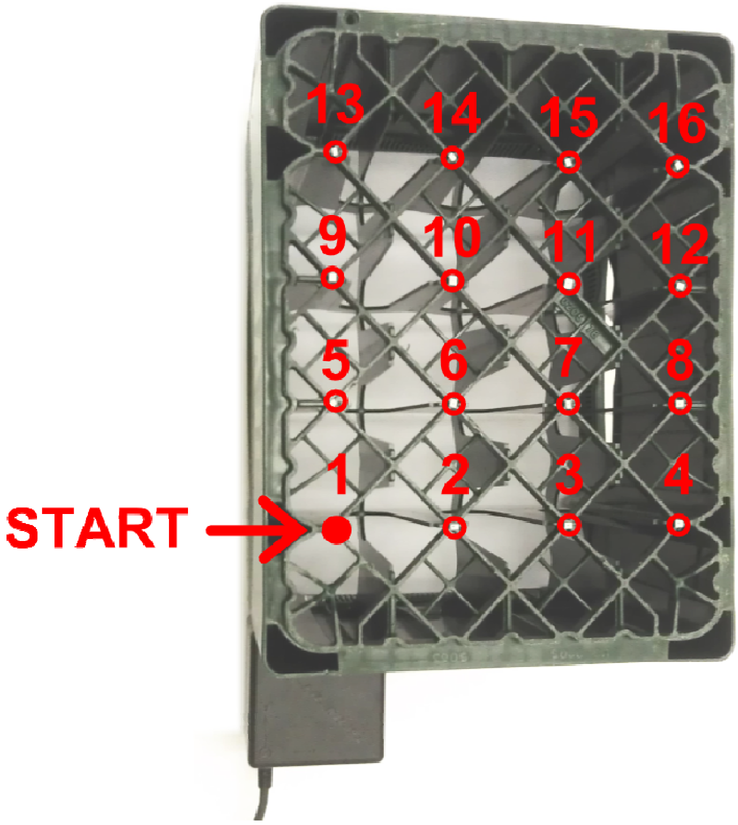


Abb. 52: Die Vorrichtung für die Kalibrierung des elektromagnetischen Feldes. In der Standardeinstellung muss der Sender am linken, unteren Rand der Bierkiste bündig platziert werden. Die Bierkiste muss mit dem Boden nach oben aufgestellt werden. Die Messung der Punkte fängt in der linken, unteren Ecke an und endet in der rechten, oberen Ecke der Bierkiste. Danach werden auf die gleiche Art und Weise die zwei nächsten Ebenen vermessen. Die Bierkisten werden dabei aufeinander gestapelt.

Entzerrung

Um eine Entzerrung der vermessenen Punkte benutzen zu können, müssen vor der Verschiebung des Senders vier Referenzpunkte in der Umgebung aufgenommen werden. Die Referenzpunkte müssen ein Koordinatensystem aufspannen und genau in der gleichen Reihenfolge abgemessen werden, wenn der Sender auf einer neuen Stelle platziert wurde. Entsprechend befindet sich im Reiter References eine Liste mit zwei Einträgen. Jeder Eintrag steht für den Namen der Gruppe, zu der der vermessene Punkt gehören soll. Erst wenn in jeder Gruppe vier Punkte aufgenommen sind, lässt sich die Koeffizientenmatrix für die Entzerrung mit dem Operator Calculate equalization berechnen (siehe Seite 94). Zu der ersten Gruppe gehören die Messpunkte, die vor der Verschiebung des Senders gesetzt sein sollen.

Zu der zweiten Gruppe gehören die Messpunkte, die nach der Verschiebung des Senders gesetzt werden. Die Anzahl der Punkte jeder Gruppe ist unter Group members zu finden (Abb. 53).

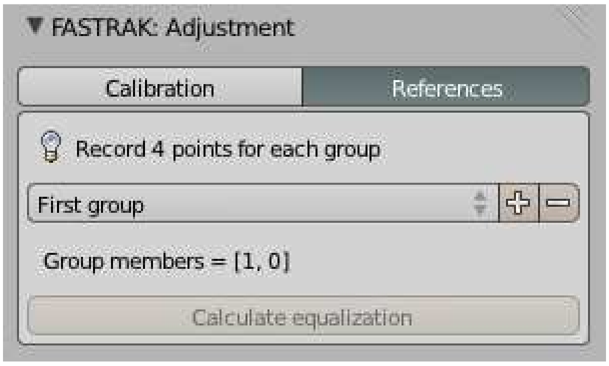


Abb. 53: Ansicht des Adjustment**-Panels zum Setzen der Referenzpunkte für die Entzerrung. Mit dem Pluszeichen werden die Punkte zu der in der Liste angezeigten Gruppe hinzugefügt. Mit dem Minuszeichen kann man die Punkte aus der Gruppe einzeln entfernen.**

Wenn die Koeffizientenmatrix berechnet wurde, wird im Kontrollpanel ein zusätzlicher Bereich Adjustment eingeblendet. Hier kann die Korrektur für die neu ankommenden Punkte ein- und ausgeschaltet werden. In der Standardeinstellung ist die Korrektur ausgeschaltet.



Abb. 54: Adjustment**-Steuerungsbereich in dem Kontrollpanel. Standardmäßig ist die Online-Korrektur der neu ankommenden Daten ausgeschaltet.**

Ermittlung des Volumens und der Masse

Die Erweiterung begrenzt sich nicht nur auf die Vermessung der topologischen Struktur von Pflanzen. Im Panel Volume (Abb. 55) kann man ein 3D-Gitterobjekt erstellen, dessen Volumen und Masse ermittelt werden sollen.



Abb. 55: Panel für die Vermessung eines Objekts, für das das Volumen und die Masse ermittelt werden sollen.

In dem Volume-Panel wird zuerst nur das 3D-Objekt vermessen und das Gitter des Objekts erstellt. Die eigentliche Berechnung des Volumens, bzw. auch der Masse wird im FASTRAK: Objekt Information-Panel durchgeführt. Der Grund dafür ist, dass das erstellte Gitter zuerst auf Korrektheit überprüft werden muss (siehe Seite 99). Wenn das Gitter alle vier Bedingungen, die dabei getestet werden müssen, erfüllt, wird der Algorithmus ein richtiges Ergebnis liefern. Danach wird im Objekt Information-Panel das Feld für die Angabe der Dichte eingeblendet. Nach der Aktualisierung des Feldes, wird zum angegebenen Wert die Masse des Objekts

ausgerechnet (Abb. 56). Die Werte für das Volumen sind in den Blender-Einheiten angegeben, deshalb muss ein passender Faktor für das Ablesen der Zieleinheiten (mm^3, cm^3, m^3 …) berücksichtigt werden. Das gleiche gilt für die Angabe der Dichte und das Ablesen der Masse. Dabei ist es wichtig zu wissen, dass der Scanner die Koordinaten in Zentimeter sendet und so als Blender-Einheiten abgespeichert werden.

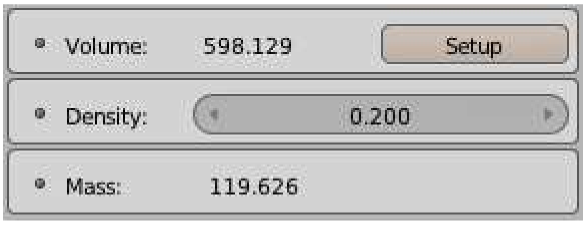


Abb. 56: Teil des FASTRAK: Objekt Information**-Panels zur Berechnung des Volumens und der Masse. Die Masse wird nach jeder Aktualisierung der Dichte neu ermittelt.**

Sammeln der Informationen über die Scanumgebung

Das Alignment-Panel (Abb. 57) dient zur Erstellung von Objekten, die die Scanumgebung beschreiben. Die Informationen die gesammelt werden, sind von der Lage des Senders abhängig. Trotzdem ist es durchaus sinnvoll bei Messungen von Pflanzen die Möglichkeit für die Aufnahme von solchen Informationen wie die geographische Richtung oder die Neigung des Bodens vorzusehen.

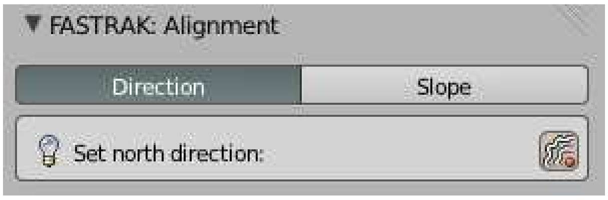


Abb. 57: Das Panel für die Erstellung der Objekte, die die Nord-Südrichtung (Direction**) und die Neigung (**Slope**) im aktuellen Messraum in Abhängigkeit zur Senderpositionierung beschreiben.**

Geographische Richtungen

Im ersten Reiter des Panels werden zwei Punkte im Einzelpunktmodus des Scanners zum Setzen der geographischen Richtungen gebraucht. Wenn die zwei Objekte für Nord und Süd gezeichnet sind, kann kein neues Objekt dieser Sorte angelegt werden. Es gibt aber hier die Möglichkeit, die Objekte mit dem Operator Clear and try again zu löschen und sie danach neu zu vermessen. Eine genauere Beschreibung der Methode zum Zeichnen der Objekte befinden sich auf der Seite 80.

Neigung gegen die XY-Ebene

Mit Hilfe des zweiten Reiters des Alignment-Panels wird die mittlere Neigung des Bodens, bzw. der vermessenen Fläche gegen die XY-Ebene des Senders ermittelt (siehe Seite 81 und 96). Die Aufnahme der Punkte wird im kontinuierlichen Modus des Scans durchgeführt. Deshalb ist das Prinzip der Steuerung der Vermessung gleich wie beim Abtasten einer Frucht (Abb. 49). Der Unterschied besteht daran, dass nach der Bestätigung der Vermessung nicht ein 3D-Gitter erstellt wird, sondern eine Punktwolke und eine Ebene, die durch die Punktwolke geführt ist

(Abb. 22). Die Punkte zusammen mit der Ebene sind im Edit Mode des Objekts zu sehen. Der berechnete Winkel ist im FASTRAK: Objekt Information-Panel in Grad eingegeben (Abb. 58). Im Addon kann man mehrere von solchen Objekten anlegen. Sie werden alle zur Gruppe Ground_slope hinzugefügt und sind in der Gruppenansicht des Outliner-Fensters zu finden.



Abb. 58: Angabe des Winkels zwischen der vermessenen Ebene und XY-Ebene des Senders im FASTRAK: Objekt Information-Panel.

Darstellung der Objektinformationen in Blender

Die Objekte die infolge der Messung entstehen, werden in Blender graphisch dargestellt. Zusätzlich bietet Blender eine Auflistung der Objektnamen in der Outliner-Ansicht. Desweiteren verfügt das Addon selbst über ein Panel, FASTRAK: Objekt Information, das der Darstellung und teilweise auch der Manipulation der Objektdaten dient.

*Outliner/Groups-**Darstellung***

Zu der graphischen Darstellung eines Objekts bietet Blender in der Outliner-Ansicht eine Liste aller in der aktuellen Session des Programms angelegten Objekte. Die Liste ist alphabetisch aufgestellt. Um die Anzahl der Objekte für die Übersichtlichkeit zu begrenzen, werden im Outliner bestimmte Filter angeboten. Bei der Standardeinstellung für den Filter (All Scenes) sind alle Objekte von allen Szenen mit Namen aufgelistet. Bei der Benutzung des Addons ist vor allem der Filter Groups zu empfehlen (Abb. 59), da die in der Erweiterung erstellten Objekte standardmäßig einer Gruppe zugewiesen sind. Somit ist es z. B. einfach, alle Objekte einer Gruppe gehören schnell zu selektieren oder ein- und auszublenden.

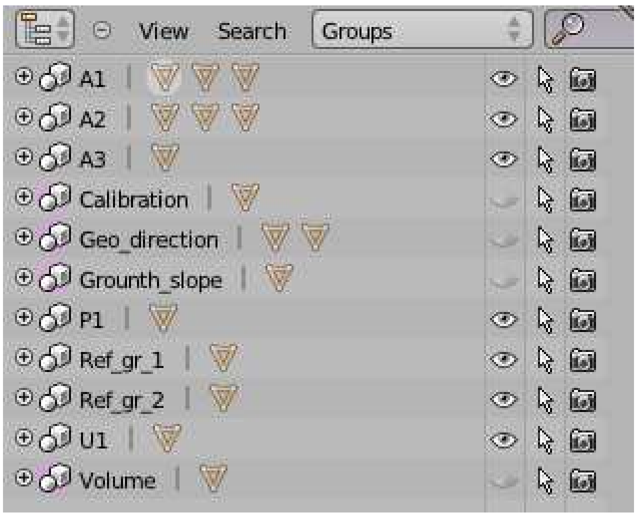


Abb. 59: Sortierung nach Gruppen in dem Outliner-Panel. Die Dreiecke symbolisieren die Objekte, die zu einer Gruppe gehören. Das geschlossene Auge bedeutet, dass alle Objekte der Gruppe in der 3D-Ansicht ausgeblendet sind. Mit dem Pfeil kann man Objekte zum Editieren freigeben oder das Editieren sperren. Mit dem Fotokamerasymbol kann man die Objekte für das Rendern sichtbar machen oder ausblenden.

Addon-Panel für Objektinformation

Die Scan Informationen werden im Panel *FASTRAK: Objekt Information* (Abb. 60), abhängig vom Typ (Tab. 11) des Objektes eingeblendet. Die Auflistung der Eigenschaften, die zu einem bestimmten Objekttyp zugeordnet sind und in diesen Panel angezeigt werden, befindet sich in Tab. 14. Für alle Objekte ist es möglich, einen Kommentar im Feld *Comment* abzuspeichern. Für jedes Internodium, das als Zylinder dargestellt wird, kann man den Durchmesser entweder für den oberen und unteren Deckel gleichzeitig oder getrennt editieren (Abb. 17). Die Angabe des Durchmessers ist hier auch für ein Internodium, das als eine Sphäre dargestellt wird, möglich. Bei einem Objekt, das für die Berechnung des Volumens vorgesehen ist, kann man in diesem Panel zum einen das Volumen berechnen lassen, und zum anderen die Dichte setzen und dadurch die Masse des Gegenstands ermitteln (Abb. 56).

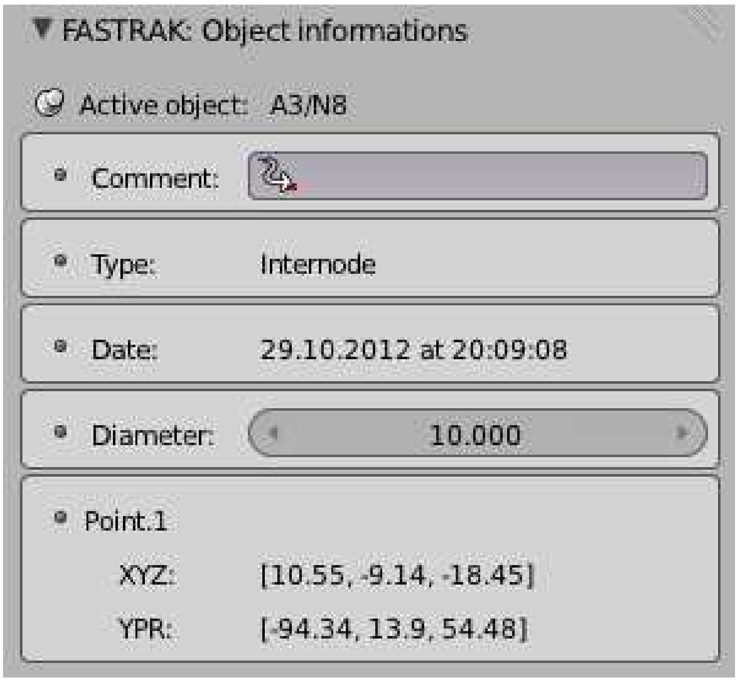


Abb. 60: Beispielansicht des *FASTRAK: Objekt Information*-**Panels. Das aktive Objekt ist ein Internodium, dessen geometrische Form eine Sphäre ist. Deshalb kann man zusätzlich zum Eingabefeld für einen Kommentar den Durchmesser der Kugel bestimmen.**

Bearbeitung der MTG-Struktur der gescannten Pflanze

Das letzte Panel des Addons ist nicht direkt mit dem Scan von Pflanzen verbunden. Es ist für die Bearbeitung der im *Topology*-Panel aufgenommenen Struktur vorgesehen. Dabei ist es möglich, die weiteren Skalierungslevels des MTG-Formats (*Multiscale Tree Graph data structure*) (Abb. 61) für die Beschreibung der Pflanzen zu benutzen. Die erste Skalenebene sind die Internodien (en. *internodes*), die Knoten des Graphen. Die Internodien werden automatisch bei der Aufnahme auf einer Achse gruppiert. Die nächste Gruppierungsebene sind die Wachstumseinheiten (en. *growth units*) und die letzte Beschreibungsebene sind die Pflanzen (en. *plants*).

Der erste Reiter *Branch* des Panels ermöglicht das Setzen einer Verzweigung im aufgenommenen Modell. Um eine Verzweigung zu setzen muss man im Feld *Basis*

node den Namen des Internodiums, bei dem die Verzweigung beginnt, hinschreiben. Alternativ kann man ein Internodium in der 3D-Ansicht oder im Outliner selektieren und dann aus der Selektion den Namen in den Feld übernehmen (). Genauso kann man das Feld Succesor axis ergänzen. Dabei ist es nicht wichtig welches Internodium der Achse selektiert wird, da in diesem Feld nur der Name der Nachfolgeachse angegeben werden muss. Wenn die zwei Felder ausgefüllt sind, kann man die Verzweigung setzten. Eine neu erstellte Verzweigung wird dann zu der Liste Branches hinzugefügt. Der zum Benutzer sichtbare Eintrag der Verzweigungsliste kann mit entfernt werden. Mit kann man alle Mitglieder der Verzweigung selektieren.

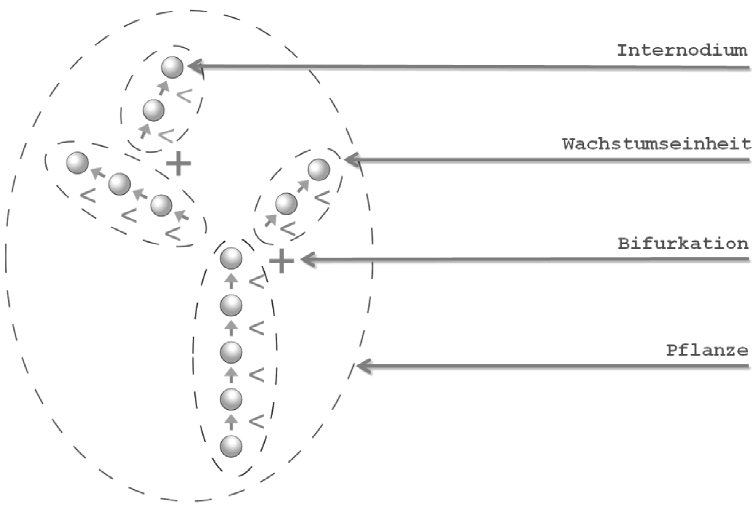


Abb. 61:Skalierung nach dem MTG-Format (Cokelaer und Christophe 2009).



Abb. 62: Branch**-Reiter des MTG-Editors.**

Der Units-Reiter dient zur Gruppierung der Internodien in die Wachstumseinheiten. Mit kann man eine neue Wachstumseinheit zu der Units-Liste hinzufügen. Wenn eine Wachstumseinheit erstellt wurde kann man mit Link node(s) die selektierten

Internodien zu der Gruppe hinzufügen und mit Unlink node(s) können die selektierte Internodien, die der in der Lise Units angezeigten Gruppe gehören, von der Gruppe entfernt werden. Die zuletzt erstellte Wachstumseinheit kann mit ⊟ entfernt werden. Dabei werden keine von den zugehörigen Internodien gelöscht (Abb. 63).

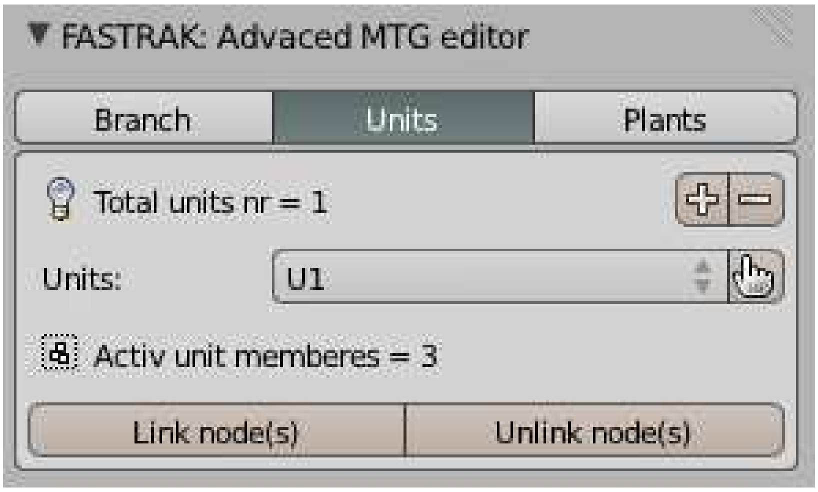


Abb. 63: Units-**Reiter des MTG-Editors.**

Der Plants-Reiter dient zur Gruppierung der Internodien in die Gruppen von einer Pflanze. Mit ⊞ kann man eine neue Wachstumseinheit zu der Plants-Liste hinzufügen. Wenn eine Pflanzengruppe erstellt wurde kann man mit Link node(s) die selektierten Internodien zu der Gruppe hinzufügen und mit Unlink node(s) können die selektierte Internodien, die der in der Lise Plants angezeigten Pflanzengruppe gehören, von der Gruppe entfernt werden. Die zuletzt erstellte Pflanzengruppe kann mit ⊟ entfernt werden. Dabei werden keine von den zugehörigen Internodien gelöscht (Abb. 64).

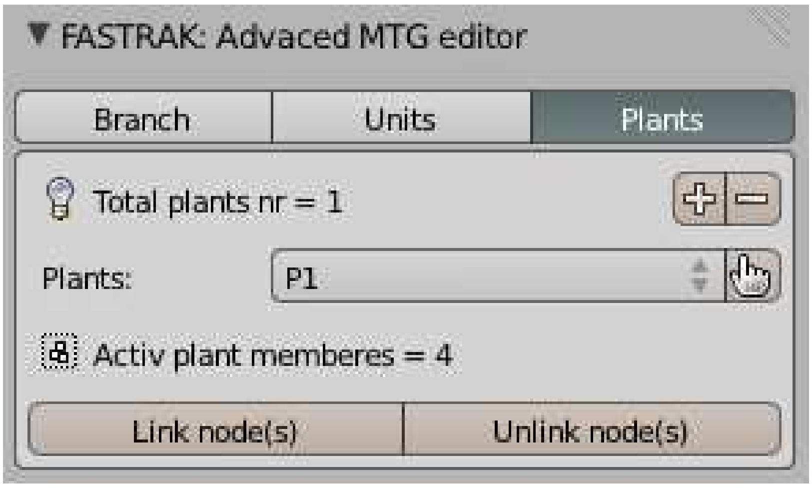


Abb. 64: Plants-**Reiter des MTG-Editors.**

Kapitel 5

Schlussbemerkungen und Ausblick

In dieser Arbeit wurde eine Erweiterung für Blender zum Scannen von Pflanzen mit dem Trackingsystem Polhemus FASTRAK entwickelt. Das neue Blender-Modul kann, neben den standardmäßigen topologischen Informationen, wie die Struktur der Pflanze, Länge und Durchmesser der Internodien, auch Blätter, Blüten und Früchte vermessen. Das Modul bietet die Möglichkeit, aus den gemessenen Daten abgeleitete Informationen, wie die Fläche der Blätter oder Blüten sowie das Volumen der Früchte, zu berechnen. Für Objekte, für die die Eigenschaft volume gesetzt ist, kann man nach der Angabe der Dichte des Objekts die Masse berechnen. Das Addon bietet auch Objekte, die zur räumlichen Orientierung in dem Modell dienen. Dazu werden die Nord-Süd-Richtung und die Neigung bezüglich des Senderkoordinatensystem vermessen. Um auch nach einer Verschiebung des Senders eine Messung fortführen zu können, wird die entsprechende Entzerrungsmatrix berechnet. Das magnetische Feld des Scanners ist empfindlich gegen elektromagnetische Störungen. Deshalb bietet das Addon eine Methode zur Kalibrierung des Feldes mit Hilfe des polynomialen Fits eines Referenzgitters. Die ankommenden Daten werden auf Wunsch des Benutzers während der Aufnahme korrigiert. Alle Daten, die zu den Objekten in Blender abgespeichert wurden, können als eine CSV-Datei exportiert werden. Die topologischen Daten können im MTG-Format exportiert werden. Dies ist zurzeit nur auf Ebene der Internodien mit Angabe der Länge, des Durchmessers und des Typs von einem Internodium möglich. Die Speicherung der abgemessenen oder das Laden schon abgespeicherten Modelle sowie auch Export der Modelldaten in andere 3D-Datenformate (obj, ply, stl …) wird von Blender bereitgestellt. Dank Blender kann das Addon sowohl unter Linux als auch unter Windows benutzt werden und ist damit betriebssystemunabhängig. Somit ist das Addon eine gute Grundlage zur Aufnahme des Pflanzenwachstums und der Struktur, das in Agronomie und Forstwirtschaft eingesetzt werden kann.

Mit dem in dieser Arbeit erstellten Blender-Addon wurde eine Reihe von Pflanzenmodellen erstellt (siehe Anhang D, Seite 139). Die Messungen wurden von verschiedenen Testpersonen durchgeführt. Dabei wurde die Nutzbarkeit der Implementierung getestet und Vorschläge für Verbesserungen und Weiterentwicklungen gesammelt. Folgende Ideen wurden aus der Benutzersicht diskutiert:

- Abtasten der Pflanze im kontinuierlichen Modus des Scanners mit gedrückter Taste des Empfängers, da es der Mehrheit der Benutzer intuitiver erscheint.

- Verkürzen der Namen der Internodien, da der Name der Gruppe in der Outliner-Liste bei der Gruppenansicht eindeutig zugeordnet ist, d.h. statt A1/N1 nur N1 für ein Internodium.
- Änderung der Namensgebung für die Internodien und Achsen, damit die Outliner-Liste die Reihenfolge der aufgenommen Objekte besser widerspiegelt (z. B. führende Nullen N0001 oder A0001).

Aus Sicht eines Programmierers sollte die nächste Entwicklungsstufe des Addons folgende Punkte berücksichtigen:

- Weiterentwicklung der Algorithmen zur Triangulierung einer Punktwolke, so dass jede geometrische Form beim genügend vielen aufgenommenen Punkten richtig trianguliert wird.
- Weitere Optimierung der Laufzeiten für die Berechnung der Triangulierungen und das Zeichnen der Objekte.
- Ausbau des MTG-Exportoperators.
- Umsetzung der oben aufgelisteten Benutzerwünsche
- Weitere Optimierung der Benutzeroberfläche
- Weitere Vermessungsmethoden nach Absprache mit den Anwendern

Anhang A - Quelltext

Beispiel 1 - Der einfache Operator

```
import bpy


class PrintNamesOperator(bpy.types.Operator):
    '''Tooltip: Prints visible objects names to the console.'''
    bl_idname = "scene.simple_names_print_operator"
    bl_label = "Simple Print Operator"

    @classmethod
    def poll(cls, context):
        return len(context.visible_objects) > 0

    def execute(self, context):
        for obj in context.visible_objects:
            print(obj.name)
        return {'FINISHED'}


def register():
    bpy.utils.register_class(PrintNamesOperator)


def unregister():
    bpy.utils.unregister_class(PrintNamesOperator)


if __name__ == "__main__":
    register()

# test call
if bpy.ops.scene.simple_names_print_operator.poll():
    bpy.ops.scene.simple_names_print_operator()
else:
    print('There are no visible objects in the scene!')
```

Beispiel 2 - Ein Panel für die Benutzung des Operators aus dem ersten Beispiel

```
import bpy


def update_count_func(self, context):
    context.scene.my_count = context.scene.my_count + 1
    return None


class PrintNamesOperator(bpy.types.Operator):
    '''Tooltip: Prints visible objects names to the console.'''
    bl_idname = "scene.simple_names_print_operator"
    bl_label = "Simple Print Operator"

    @classmethod
    def poll(cls, context):
        return len(context.visible_objects) > 0


    def execute(self, context):
        for obj in context.visible_objects:
            print(obj.name)
        return {'FINISHED'}
```

```
class SimplePrintProperty(bpy.types.PropertyGroup):
    show = bpy.props.BoolProperty(name = "Show Operator",
                                  description = "Show/hide the operator
button.",
                                  default = True,
                                  update = update_count_func)


class SimplePrintPanel(bpy.types.Panel):
    bl_label = "Simple Print Panel"
    bl_idname = "OBJECT_PT_simple_panel"
    bl_space_type = "PROPERTIES"
    bl_region_type = "WINDOW"
    bl_context = "scene"

    bpy.types.Scene.my_count = bpy.props.IntProperty(default = 0)

    def draw(self, context):
        layout = self.layout
        col = layout.column()
        col.prop(context.scene.my_simpe_print, "show")
        box = col.box()
        if context.scene.my_simpe_print.show:
            box.operator("scene.simple_names_print_operator")
        else:
            box.label(" Simple print disabled.", icon = "INFO")
            box.label(" You have pressed the button %s times." %
context.scene.my_count,
                        icon = "LAMP")


def register():
    bpy.utils.register_class(SimplePrintProperty)
    bpy.utils.register_class(PrintNamesOperator)
    bpy.utils.register_class(SimplePrintPanel)

    bpy.types.Scene.my_simpe_print = bpy.props.PointerProperty(type =
SimplePrintProperty)


def unregister():
    bpy.utils.unregister_class(SimplePrintProperty)
    bpy.utils.unregister_class(PrintNamesOperator)
    bpy.utils.unregister_class(SimplePrintPanel)
```

Beispiel 3 – Erweiterung des zweiten Beispiels zu einem Addon für Blender

```
#====================== BEGIN GPL LICENSE BLOCK ======================
#
#  This program is free software; you can redistribute it and/or
#  modify it under the terms of the GNU General Public License
#  as published by the Free Software Foundation; either version 2
#  of the License, or (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to the Free Software Foundation,
#  Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
#
#======================= END GPL LICENSE BLOCK =======================
```

```
bl_info = {"name": "Simple Print Names Addon",
           "author": "Katarzyna Wasilczuk",
           "version": (1, 0),
           "blender": (2, 6, 2),
           "location": "Properties Window > Scene",
           "description": "Text abouth this addon.",
           "warning": "e.g. dependieces ...",
           "wiki_url": "",
           "tracker_url": "",
           "category": "3D View"}

import bpy


def update_count_func(self, context):
    context.scene.my_count = context.scene.my_count + 1
    return None


class PrintNamesOperator(bpy.types.Operator):
    '''Tooltip: Prints visible objects names to the console.'''
    bl_idname = "scene.simple_names_print_operator"
    bl_label = "Simple Print Operator"

    @classmethod
    def poll(cls, context):
        return len(context.visible_objects) > 0

    def execute(self, context):
        for obj in context.visible_objects:
            print(obj.name)
        return {'FINISHED'}


class SimplePrintProperty(bpy.types.PropertyGroup):
    show = bpy.props.BoolProperty(name = "Show Operator",
                                  description = "Show and hide the operator
button.",
                                  default = True,
                                  update = update_count_func)


class SimplePrintPanel(bpy.types.Panel):
    bl_label = "Simple Print Panel"
    bl_idname = "OBJECT_PT_simple_panel"
    bl_space_type = "PROPERTIES"
    bl_region_type = "WINDOW"
    bl_context = "scene"

    bpy.types.Scene.my_count = bpy.props.IntProperty(default = 0)

    def draw(self, context):
        layout = self.layout
        col = layout.column()
        col.prop(context.scene.my_simpe_print, "show")
        box = col.box()
        if context.scene.my_simpe_print.show:
            box.operator("scene.simple_names_print_operator")
        else:
            box.label(" Simple print disabled.", icon = "INFO")
        box.label(" You have pressed the button %s
times."%context.scene.my_count,
                  icon = "LAMP")


def register():
    bpy.utils.register_module(__name__)

    bpy.types.Scene.my_simpe_print = bpy.props.PointerProperty(type =
SimplePrintProperty)


def unregister():
    bpy.utils.unregister_module(__name__)
```

Beispiel 4: Serielle Kommunikation mit Hilfe des Moduls *pyserial*

```
import serial
from sys import platform
from re import search


if platform == 'win32':
    ser = serial.Serial('COM16')

if search('linux', platform):
    ser = serial.Serial('/dev/ttyS0')

if not ser.isOpen():
    ser.open()
else:
    print('\nThe connection is opened on port = %s\n'%ser.port)

ser

ser.baudrate = 115200
ser.timeout = 0.1

ser

ser.write(bytes('S', 'latin1'))
ser.readline()
```

Anhang B – Anwendungsverzeichnis

Lokalisation des Verzeichnisses

Für freigeben des Verzeichnisses kann man die *Arbeitsgruppe für Pflanzenmodellierung* von Prof. Dr. Winfried Kurth an der *Abteilung Ökoinformatik, Biometrie und Waldwachstum* kontaktieren.

```
'svn+ssh: ...\3D_SCANN_ANWENDUNG'
```

Inhalt der Anwendungsverzeichnis

Softwareverzeichnis

```
'svn+ssh: ...\3D_SCANN_ANWENDUNG\software\windows'
```

- fastrak_cd (Die CD zu dem Polhemus FASTRAK Trackingsystem, mit Treiber, Software und Benutzerhandbuch)
- piaffdigit (Das Programm PiafDigit und eine Beispieldatei von Dr. Peter Srovy)
- blender-2.62-release-windows32.exe (3D Modellierungssoftware)
- python-3.2.2.msi (Windows Installer für Python 3.2)
- pyserial-py3k-2.5.win32.exe (Python-Modul für serielle Kommunikation)
- numpy-1.6.2-win32-superpack-python3.2.exe (Python-Modul für numerische Berechnungen)
- scipy-0.11.0rc1-win32-superpack-python3.2.exe (Wissenschaftliche Werkzeuge für Python)

```
'svn+ssh: ...\3D_SCANN_ANWENDUNG\software\linux'
```

- ubuntu-11.10-desktop-i386.iso (Installation-CD für Ubuntu 11.10)
- blender-2.62-linux-glibc27-i686.tar.bz2 (3D Modellierungssoftware)
- Python-3.2.tgz (Python-Quellen)
- pyserial-2.5.tar.gz (Python-Modul für serielle Kommunikation)
- numpy-1.6.2.tar.gz (Python-Modul für numerische Berechnungen)
- scipy-0.11.0rc2.tar.gz (Wissenschaftliche Werkzeuge für Python)
- pygame-1.9.1release.tar.gz (Python-Modul für Programmierung der Spiele)

Verzeichnis der Erweiterung

```
'svn+ssh: ...\3D_SCANN_ANWENDUNG\addon\fkscn
```

Das Verzeichnis fkscn ist das eigentliche Verzeichnis der Erweiterung für Blender. Damit das Addon über die Benutzeroberfläche verwalten werden kann, muss er sich im Blender-Verzeichnis für die Erweiterungen befinden (Kapitel 1, Seite 34 und Kapitel 2, Seite 49).

Anhang C – Installationen

Windows-Installer für *Python 3.2*, *pyserial*, *numpy* und *scipy*

Die Installer für Python 3.2, pyserial, numpy und scipy können entweder aus dem Internet heruntergeladen werden oder direkt aus dem Anwendungsverzeichnis im SVN-Archiv (Anhang B, Seite 137) kopiert und gestartet werden. Als erster muss Python 3.2 mit python-3.2.2.msi installiert werden (Abb. 65). Der Pfad zum Installationsverzeichnis muss in die Windows-Umgebungsvariable angetragen werden. Danach kann man Python in der Systemkonsole mit dem Befehl python starten. Anschließend kann man die Installationen von pyserial (pyserial-py3k-2.5.win32.exe), numpy und scipy durchführen (Abb. 66). nur dann erfolgreich verlaufen, wenn Python 3.2 auf dem Rechner installiert ist und die Umgebungsvariable path für Python gesetzt ist.

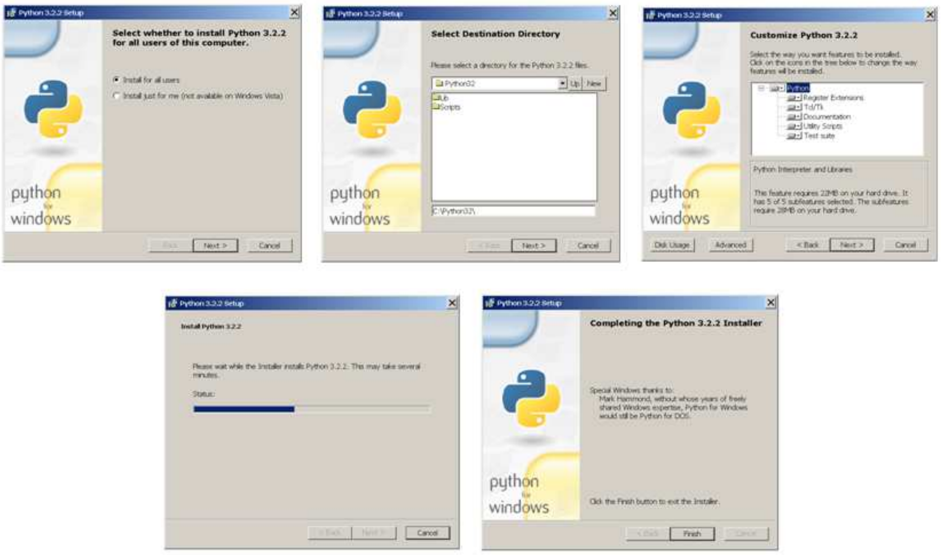


Abb. 65: Ansichten des Windows-Installers für Python 3.2**.**

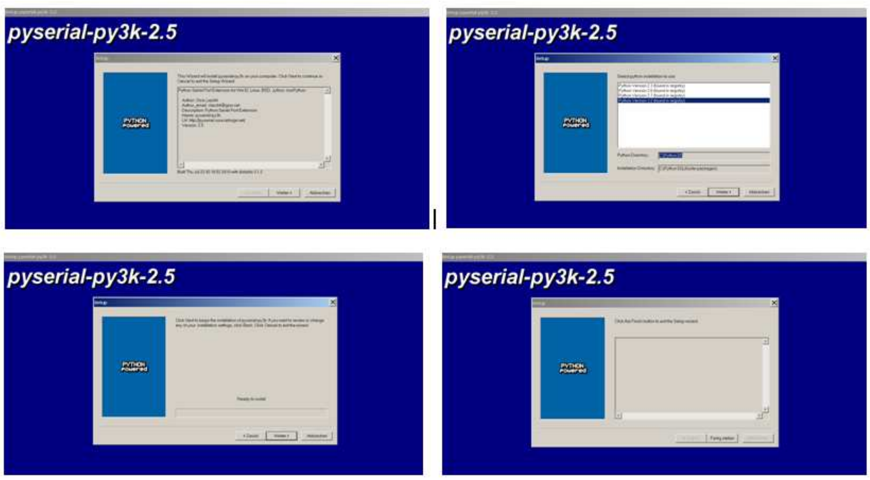


Abb. 66:Ansicht der Installation des Moduls pyserial unter Windows. Die Installers für die Module numpy und scipy sehen analog aus. Dabei ist zu beachten, dass zuerst das numpy Modul installiert werden muss, erst danach kann man das scipy-Modul installieren.

Anhang D – Messungen

Ergebnis der Vermessung einer Erdbeere

Parameter	Wert
Anzahl der Personen bei der Vermessung	1
Dauer	45 min
Anzahl der abgemessenen Internodien	92
Fläche der Blätter	246,7 cm^2
Volumen der Früchte	13,1 cm^3
Volumen des Topfs	357,5 cm^3

Ergebnis der Vermessung einer Mohnblüte





Parameter	Wert
Anzahl der Personen bei der Vermessung	1
Dauer	5 min
Anzahl der abgemessenen Internodien	11
Fläche der Blüte	72,8 cm^2

Ergebnis der Vermessung einer Clementine





Parameter	Wert
Anzahl der Personen bei der Vermessung	2
Dauer	3h
Anzahl der abgemessenen Internodien	291
Fläche der Blätter	791,1 cm^2
Volumen des Topfs	1565,9 cm^3

Literaturverzeichnis

Blender Foundation. *Blender Documentation Contents - Blender 2.62.2 - API documentation.* 2012. http://www.blender.org/documentation/blender_python_api_2_62_2/ (Zugriff am 10. 09 2012).

—. *blender.org - Blender Foundation.* 2012. http://www.blender.org/blenderorg/blender-foundation/ (Zugriff am 10. 09 2012).

—. *blender.org - Home.* 2012. http://www.blender.org/ (Zugriff am 10. 09 2012).

—. *Dev:2.5/Py/Scripts/Cookbook/Code snippets/Interface - BlenderWiki.* 2012. http://wiki.blender.org/index.php/Dev:2.5/Py/Scripts/Cookbook/Code_snippets/Interface (Zugriff am 10. 09 2012).

—. *Doc:2.6/Manual/Data System/The Outliner - BlenderWiki.* 2012. http://wiki.blender.org/index.php/Doc:2.6/Manual/Data_System/The_Outliner (Zugriff am 26. 10 2012).

—. *Doc:2.6/Manual/Interface/Panels - Blender Wiki.* 2012. http://wiki.blender.org/index.php/Doc:2.6/Manual/Interface/Panels (Zugriff am 26. 10 2012).

—. *Doc:2.6/Manual/Modeling/Objects/Selecting - BlenderWiki.* 2012. http://wiki.blender.org/index.php/Doc:2.6/Manual/Modeling/Objects/Selecting (Zugriff am 26. 10 2012).

—. *Gotchas - Blender 2.62.0 - API documentation.* 2012. http://www.blender.org/documentation/blender_python_api_2_62_0/info_gotcha.html (Zugriff am 18. 09 2012).

—. *Index of /release/Blender2.62/.* 2012. http://download.blender.org/release/Blender2.62/ (Zugriff am 10. 09 2012).

—. *Operator(bpy_struct) - Blender 2.62.2 - API documentation.* 2012. http://www.blender.org/documentation/blender_python_api_2_62_2/bpy.types.Operator.html?highlight=operator#bpy.types.Operator (Zugriff am 10. 09 2012).

—. *UILayout(bpy_struct) - Blender 2.62.2 - API documentation.* 2012. http://www.blender.org/documentation/blender_python_api_2_62_2/bpy.types.UILayout.html?highlight=layout#bpy.types.UILayout.separator (Zugriff am 10. 09 2012).

Cokelaer, Thomas, und Pradal Christophe. *MTG User Guide.* 2009. http://openalea.gforge.inria.fr/doc/vplants/mtg/doc/html/user/index.html (Zugriff am 11. 06 2012).

Donès, N., B. Adam, und H. Sinoquet. *PiafDigit - software to drive a Polhemus Fastrak 3D digitiser and for the acquisition of plant architecture. Version 1.0.* Clermont-Ferrand: UMR PIAF INRA-UBP, 2006.

Farin, Gerald, und Dianne Hansford. *Lineare Algebra: Ein geometrischer Zugang.* Springer-Verlag, 2003.

Hübner, Ralph. *Der Unterschied zwischen keiner Drehung und einer Drehung um 2π.* Wesel, 2011.

Ikits, Milan, J. Dean Brederson, Charles D. Hansen, und John M. Hollerbach. „An Improved Calibration Framework for Electromagnetic Tracking Devices.“ *Proceeding, VR '01 Proceedings of the Virtual Reality 2001 Conference (VR'01)*, 2001: 69-77.

Kindratenko, Volodymyr. „Calibration of electromagnetic tracking devices.“ *Virtual Reality: Research, Development, and Applications*, 1999: 139-150.

Landau, L. D., und E. M. Lifschitz. *Lehrbuch der Theoretischen Physik. Band 1. Mechanik.* Harri Deutsch, 1990.

Liechti, Chris. *pySerial v2.5 documentation.* 2010. http://pythonxy.googlecode.com/hg-history/b2191eb18b81d78629fea64c15a42d4fb7847efb/src/python/pyserial/DOC/index.html (Zugriff am 11. 09 2012).

—. *Python Serial Port Extension - Browse /pyserial/2.5 at SourceForge.net.* 2010. http://sourceforge.net/projects/pyserial/files/pyserial/2.5/ (Zugriff am 2012).

Mauritczat, Marco. *Proseminar Simulation und Virtuelle Realität in der Medizin: Navigation.* Karlsruhe: Karlsruher Institut für Technologie, 2003.

Merz, Markus. *Visualisierung und gezielte Selektion von Isoflächenkomponenten.* Stuttgart: Universität Stuttgart, 2001.

Nischwitz, Alfred, Max Fischer, und Peter Haberäcker. *Computergraphik und Bildverarbeitung.* Friedr. Vieweg & Sohn Verlag, 2007.

Nokia. *qmake Project Files | Documentation | Qt Developer Network.* 2012. http://qt-project.org/doc/qt-4.8/qmake-project-files.html (Zugriff am 17. Juni 2012).

—. *Qt - A cross-platform application and UI framework.* 2012. http://qt.nokia.com/products/ (Zugriff am 17. Juni 2012).

Numpy developers. *Scientific Computing Tools For Python - Numpy.* 2012. http://numpy.scipy.org/ (Zugriff am 11. 09 2012).

Polhemus. *3D Space Fastrak User Manual.* Colchester, Vermont U.S.A., 2009.

Pygame Developers. *Downloads - pygame - python game development.* 2012. http://www.pygame.org/download.shtml (Zugriff am 12. 09 2012).

—. *Pygame Readme.* 2012. http://www.pygame.org/readme.html (Zugriff am 12. 09 2012).

Python Developers. *SciPy: Scientific Library for Python - Browse /scipy/0.11.0rc1 at SourceForge.net.* 2012. http://sourceforge.net/projects/scipy/files/scipy/0.11.0rc1/ (Zugriff am 12. 09 2012).

QextSerialPort . *qextserialport - Serial port library for Qt2, Qt3, Qt4 and Qt5 - Google Project Hosting.* 2012. http://code.google.com/p/qextserialport/ (Zugriff am 17. Juni 2012).

Rosendahl, Andreas P. *Anbindung multidimensionaler Eingabegeräte für VR- und AR-Anwendungen.* Universität Koblenz, 2003.

Scipy developers. *SciPy.* 2012. http://www.scipy.org/ (Zugriff am 11. 09 2012).

Siebert, Dirk. *Ein Meßsystem zur präoperativen Planung und intraoperativen Kontrolle von Dysgnathieoperationen.* Berlin, 2002.

Simple DirectMedia Layer Developers. *Simple DirectMedia Layer.* 2012. http://www.libsdl.org/ (Zugriff am 12. 09 2012).

SourVR. *Polhemus Fastrak_Magnetic_SouVR.* 2008. http://en.souvr.com/product/200712/242.html (Zugriff am 29. 10 2012).

Tegtmeier, André. *Augmented Reality als Anwendungstechnologie in der Automobilindustrie.* Magdeburg: Otto-von-Guericke-Universität, 2006.

The Python Software Foundation. *Overview - Python v3.2 documentation.* 2011. http://docs.python.org/release/3.2/ (Zugriff am 09. 09 2012).

—. *Python 3.2 Release.* 2011. http://www.python.org/download/releases/3.2/ (Zugriff am 10. 09 2012).

The Scipy community. „scipy.spatial.Delaunay - SciPy v0.11 Reference Guide (DRAFT).“ 2012. http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html#scipy.spatial.Delaunay (Zugriff am 22. 10 2012).

Ubuntu. *Ubuntu 11.10 (Oneiric Ocelot).* 2011. http://releases.ubuntu.com/oneiric/ (Zugriff am 10. 09 2012).

UMR PIAF INRA-UBP. *INRA PIAF English.* 2012. http://www1.clermont.inra.fr/piaf/index_eng.php (Zugriff am 09. 09 2012).